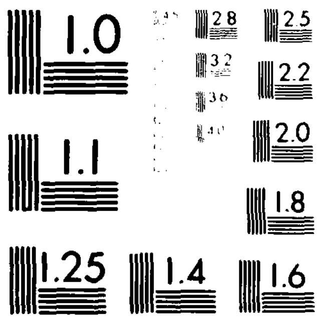
AD-A089 733 SYRACUSE UNIV NY SCHOOL OF COMPUTER AND INFORMATION --ETC F/G 9/2
MATHEMATICAL SEMANTICS FOR HIGHER ORDER PROGRAMMING LANGUAGES.(U)
JUL 80 L E SANCHIS F49620-79-C-0016
UNCLASSIFIED AFOSR-TR-80-0759 NL

1 OF 1
AD A
089733

END
DATE
FILMED
11-80
DTIC



1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

AFOSR-TR-80-0759

(11)

(18) (19)

LEVEL II

AD A089733

(15) FINAL REPORT F49620-79-C-0016

(9) Final rept. 1 Jun 79-31 May 80

(6) TITLE: MATHEMATICAL SEMANTICS FOR HIGHER ORDER PROGRAMMING LANGUAGES.

(11) Jul 80 (12) 66

(10) AUTHOR: LUIS E. SANCHIS
~~School of Computer~~ and Information Science
Syracuse Univ~~ersity~~
~~Syracuse~~, N.Y. 13210

(16) 2304 (17) A2

PREPARED FOR: AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Bolling AFB
D.C. 20332

80 9 25 051

410176

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 80-0759	2. GOVT ACCESSION NO. AD-A089 733	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MATHEMATICAL SEMANTICS FOR HIGHER ORDER PROGRAMMING LANGUAGES		5. TYPE OF REPORT & PERIOD COVERED Final
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Luis E. Sanchis		8. CONTRACT OR GRANT NUMBER(s) F49620-79-C-0016
9. PERFORMING ORGANIZATION NAME AND ADDRESS Syracuse University School of Computer & Information Science Syracuse, NY 13210		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, Washington, DC 20332		12. REPORT DATE July 1980
		13. NUMBER OF PAGES Seven
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Programming languages, Procedure definition, copy rule, Lambda calculus, Church-Rosser property, Nondeterministic computability, Reflexive domains, Reductional semantics, Structural semantics

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Complete operational and mathematical semantics are presented for a higher order applicative algorithmic language (BAL). Both semantics involve partially ordered domains closed under limits of convergent sequences. Procedure calls are formalized via lambda calculus reductions, or copy rule. Evaluations involve a more general form of computability described as nondeterministic computability. The mathematical semantics is obtained via embeddings in reflexive domains. Both semantics are proved to be equivalent, and several applications are given. The adequacy of the copy rule is proved in standard

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract cont.

situations where computability is deterministic. Several examples are presented.

INTRODUCTION

This report covers research performed under AFOSR sponsorship from June 1st, 1979 until May 31st, 1980. A number of results have been obtained, which have been organized in one substantial paper entitled: Algorithmic Language Semantics and Nondeterministic Computability. It is expected that this paper will be submitted for publication, and a copy is included as part of this report. The paper will be referred to as ALS.

Some ideas and results included in ALS were actually obtained during the early stage of this project, under RADC sponsorship. But only one result has been reported previously, namely the proof of the Church-Rosser Property which was included in the final report to RADC, and appears also in section 1 of ALS. All other results are reported here for this first time.

MAIN RESULTS

The central topic of the project was the problem of mathematical semantics for higher order programming languages. It was intended to apply lattice-theoretical methods, and a considerable amount of information about these structures has been collected under the former sponsor. The proposal to AFOSR was explicitly committed to use a particular type of structures called reflexive domains. At the time we began the research reported here we decided to apply the method to a non-trivial language involving several higher order principles.

We chose a basic language specially designed to clarify the essential problems of higher order languages. The language was called BAL, and a complete description is given in section 1 of ALS.

The language involves a few operators, which are implemented in a natural way, avoiding restrictions intended to make the language more 'practical' or 'efficient'. The only restrictions are those required by the meaning of the operators which are necessary to prove the fundamental properties of the language. The idea is to isolate the semantical problems, and not to contribute another language to the computer daily practice. Still BAL is quite a powerful language in which recursive programming is possible, even if such a feature is not explicitly allowed.

The most severe restriction in our approach is that BAL is an applicative language. There is no doubt it would have been of greater significance to consider a sequential language containing procedure definitions and assignment statements. But we do not think we are ready yet for such an application. At any rate applicative languages are important, and while they simply dissolve the dynamical structure of sequential languages they also exhibit some of their typical problems. The experience we have obtained with BAL should prove useful when the time come to consider other types of languages.

The language BAL is a truly higher order language, and contains procedure definitions in the form of lambda abstraction. Two other important features appear in BAL. There is a branching operator that responds to the usual conception in computer languages, and there is a particular operator which we call a ground operator. It is not operational, in the sense that no reduction rule is associated with this construct. Its role is to isolate a program in a given context, and it is essential to perform substitutions of programs. It seems to be related to assignment statements in sequential languages.

The syntax of BAL is described using well-known techniques of the lambda calculus. Actually BAL is an extension of the lambda calculus, in which the usual notions of redex and reduction generalize in a natural way. We prove that the reduction relation in BAL satisfies the Church-Rosser Property, thus generalizing the well-known result in the lambda calculus. This property, whose proof is given in section 1 of ALS, is essential for the semantics proposed in in the next section. The technique used in the proof presents some interest. Residuals are completely avoided, and only structural inductions are used.

The operational, or reductional, semantics for BAL is given in section 2, while the mathematical, or structural, semantics appears in sections 3 and 4, where the equivalence between both semantics is proved. The operational semantics formalizes the copy rule, embodied in type II reductions, and also the obvious meaning of the branching operator, embodied in type III reductions. This treatment of the copy rule presents some peculiarities requiring explanation.

Normally the copy rule is suppossed to be applied a number of times, until a program is obtained which can be evaluated without further procedure invocations. We use rather the copy rule to generate a sequence of programs, each one of them being evaluated assuming that any procedure call has undefined arguments. This produces a sequence of partial values, which on the basis of the Church-Rosser Property can be shown to be convergent to some limit. That limit is defined to be the output value of the program.

The advantage of this method is the way it relates to the mathematical semantics defined in section 3. Essentially it makes it possible to prove in a fairly natural manner the equivalence of both semantics. But it need justification, and several sections in ALS are devoted to clarify this matter. First of all this conception requires an adjustment of the current notion of computability, so we introduce a more general conception called nondeterministic computability. This idea is discussed in some detail in the introduction of ALS. Second we must show that our characterization of the copy rule actually agrees with the usual meaning, at least in some standard situations. This is done in section 6 of ALS, where we prove that whenever the primitive operators satisfy some rather natural conditions, then our definition of the copy rule agrees with the usual one in language semantics. This is equivalent to saying that in such cases the computation is actually deterministic.

Our conception of the copy rule imposes a number of mathematical restrictions. For instance we must assume that values are partially ordered in such a way that limits of convergent sequences exist. This is not in fact a restriction, for any set of objects can be considere partially ordered by the identity relation. So our theory is actually quite general and includes the usual forms of computability. It is worth mentioning here that our conception of the copy rule was anticipated in our proposal to AFOSR under the name of principle of copy rule completeness.

The main result of this project is the equivalence between the operational and mathematical semantics. Several applications are given in section 5, which actually depend on the structural character of the mathematical semantics. For instance we prove that computability in BAL is closed under substitution.

CONCLUSIONS

While some of the proposed objectives of the project have been obtained others will require further effort. The project was conceived as a highly theoretical endeavor, involving some sophisticated mathematics, but is was also expected that some aspects of real life computer languages would be clarified. The language BAL includes principles occurring in computer practice, and our construction reveals some of the mathematical complexities underlying their applications.

The next step should be the extension of this technique to sequential languages. This is by no means a trivial extension, and will require a complete formalization of the copy rule in the dynamical environment which is proper for sequential languages. On the other hand we may expect the mathematical semantics to be essentially the same given for BAL via embeddings in reflexive domains.

The results obtained in the paper were obtained by the principal investigator. Valuable cooperation was provided by the graduate assistant George Mouradian, who has been associated for several years with this project.

PREPRINT

ALGORITHMIC LANGUAGE SEMANTICS AND NONDETERMINISTIC COMPUTABILITY

Luis E. Sanchis

School of Computer and Information Science
Syracuse University
1980

CONTENTS

0. INTRODUCTION

0.1 During the last ten years or so a considerable amount of attention has been given to the semantics of algorithmic languages, with emphasis on the so called denotational or mathematical semantics. To deal with this problem a rather abstract theory has been developed, involving complete lattices and continuous operations. Although the main ideas were introduced by Scott in connection with the lambda calculus, and some logicians have shown interest, it was among computer scientists that the new approach was taken with definite enthusiasm.

This form of semantics is concerned with languages containing higher order definitions of functions and procedures. Any semantics has to make explicit the basic computability of the different operators entering the language. It was in dealing with procedures that the traditional notion of computability was found to be defficient. In this theory computations deal with finite discrete objects, symbols or numbers, and it was Scott's crucial insight that computability of functions involved rather an approximation process, which should be described in terms of partial orders and continuity.

Essentially the lattice approach is an attempt to provide a more general theory of computability, which in turn should be the basis for the semantics of higher order languages. But this theory never came to be fully developed, although Scott has discussed some important features (see[6]).

The most important contribution in the direction of computability was the approximation theorem proved in Wadsworth [8], which actually is presented in terms of semantics. We attempt in this paper to formalize a more general notion of nondeterministic computability via an algorithmic language called BAL. We prove that the denotational semantics anticipated by Scott is actually complete, relative to this notion of computability.

0.2 A computation involves an effective process that generates a sequence of states, a crucial property being the possibility that such a sequence may be infinite. Deterministic computability is characterized by the following fundamental assumption: *an infinite computation is undefined and produces no output.* In this conception computability requires essentially some terminating machinery, or final states, otherwise no output is ever possible.

In order to obtain nondeterministic computability we must allow for infinite computations producing output. But this conception must be refined. Obviously we do not want to call a computation any situation in which some output is generated by an infinite process, even if the process is effective in some sense. What we want is that the output should be determined as a limit of the process. More precisely, the infinite process should produce a sequence of partial values, each one by a finitary computation, and the final output should be the limit in the sense that it contains all the partial values, and nothing else.

A well known example of this form of computability is enumeration reducibility (see [2] and [4]). If Φ is an enumeration operator, and $\Phi(A) = B$, where A and B are sets of numbers, then B is generated by an infinite process which produces an infinite sequence of finite sets, B being the union of all such partial values. The process is infinite even if A and B are finite sets, for no terminating machinery exists. If B is finite then after some stage the process will produce partial values equal to B, but B is determined as the output only as a limit, and not by any of the partial values.

0.3 In formalizing these ideas we follow closely Scott [6], but note that our motivation is given in terms of the basic notion of nondeterministic computability. We shall assume the domain of values to be partially ordered. A limit is understood as a least upper bound in the usual sense in lattice theory. We need only to assume that limits exist for directed subsets.

The possibility of an undefined computation must enter explicitly in our theory, so we introduce a special undefined value, characterized as the least element in the ordering. This is sufficient if the operations are monotonic.

A sequence of partial values may present conflicting results (say true and false), so the limit in such cases must be an inconsistent or overdefined element. It is characterized as the greatest element in the partial order.

These assumptions are sufficient to introduce functions which are computable in a nondeterministic sense, but are not deterministically computable. For example consider a function f such that $f(x) = 0$ if x is the undefined value, and otherwise, $f(x) =$ the overdefined value. It is possible that the input x = undefined be given via some infinite deterministic computation, so no deterministic computation of $f(x) = 0$ is possible. On the other hand a nondeterministic computation is possible, simply by generating partial values all equal to 0, and of course converging to 0.

A final restriction must be imposed. We want the computable operations to be closed under substitution. Since output values are obtained as limits of partial values, and will enter as inputs of computations in the same way, we must require that the operations be continuous. We must point that this is in fact a crucial restriction (see [6]) that makes possible the structural semantics developed in sections 3 and 4.

0.4 Our construction is built around the algorithmic (or programming) language BAL. This is a type free applicative lambda calculus,with a conditional operator and a ground operator. The language is interpreted in basic structures consisting of partially ordered domains. A direct interpretation is possible only for terms (i.e. programs) in normal form. For general programs the interpretation is given by a reductional semantics that essentially describes the underlying computability assumed by the theory, which is of course nondeterministic computability in the sense discussed above.

The main result is the existence of a structural semantics which is equivalent to the reductional, and it is independent of any notion of computability. The semantics was originally introduced by Scott [5] for a simple lambda calculus, and a proof of equivalence for this calculus was given by Wadworth [8]. We extend their results to the language BAL.

Our use of the lambda calculus departs from the traditional approach, as given for instance in [1] chapter 3. We do not encode numerical values in terms of the calculus, but rather consider all values as denotations of variables and constants. The conditional operator works on the same assumption, and cannot be defined using application and abstraction.

0.5 Research presented in this paper has been supported by the U.S. Air Force under contracts RACD F 30602-76-C-0325 and AFOSR F 49620-79-C-0016.

1. THE LANGUAGE BAL

1.1 In this section we describe a formal language, called Basic Algorithmic Language, or simply BAL. It is essentially a lambda calculus extended with a conditional operator and a ground operator.

The language contains a countable number of formal variables: $v_1, v_2, \ldots$, the symbol Ω, and a set of constants symbols including at least the symbols **tt** and **ff**. Other symbols are used to describe the basic operations, as explained below. Letters u,v and w are used to denote variables. If X and Y are expressions in BAL then $X \equiv Y$ means that they are formally identical.

The fundamental structure of BAL are programs or terms as they are usually called in applicative languages. A *term* is defined inductively as follows:

T1: Variables, the symbol Ω, and constants are terms, also called atomic terms.

T2: If X and Y are terms then (XY) is a term, also called an applicative term.

T3: If X is a term and v is a variable then λvX is a term,also called an abstraction term.

T4: If X, Y and Z are terms then $(X \supset Y,Z)$ is also a term, also called a conditional term.

T5: If X is a term and i is a nonnegative integer then $\{X\}_i$ is a term,also called a ground term.

Letters U,V,W,X,Y,Z will denote terms. We follow the usual notation in combinatory logic and lambda calculus (see [1]) . In particular we assume the definition of the notion: the variable v occurs free (or bound) in the term X. And also the definition of the notion: the term Y is free for the variable v in the term X.

If X and Y are terms and v is a variable then the notation $[Y/v]X$ denotes the result of replacing all free occurrences of v in the term X by the term Y. Note that no change of bound variables is assumed. Hence in case $u \not\equiv v$ then $[Y/v]\lambda uX \equiv \lambda u[Y/v]X$.

We shall say that the *variable* u *is strictly free for the variable* v *in the term* X in case that $[v/u][u/v]X \equiv X$. This is actually equivalent to saying that u is free for v in X and in case $u \not\equiv v$ then u does not occur free in X. If u is strictly free for v in X then for any term Y the following relation holds: $[Y/u][u/v]X \equiv [Y/v]X$.

We assume the usual conventions to avoid writing too many parentheses. Hence parenthese are replaced by associating to the left. In this way the expression XYZ stands for the term ((XY)Z). It follows that any term X has a unique expression in the form $X_1 \ldots X_n$, $n \geq 1$, where X_1 is not an applicative term.

We introduce also the following conventions. A term of the form $\lambda vX_1X_2\ldots X_n$, $n \geq 1$, is understood as $X_1'X_2\ldots X_n$ where $X_1' \equiv \lambda vX_1$. And a term of the form $[Y/v]X_1X_2\ldots X_n$, $n \geq 1$, is understood as $X_1'X_2\ldots X_n$ where $X_1' \equiv [Y/v]X_1$.

1.2 We shall need a number of properties of the substitution operator, which in some cases will be assumed without proof. For example it is obvious that in case $v \not\equiv u$, v does not occur free in W and u does not occur free in V then $[V/v][W/u]X \equiv [W/u][V/v]X$.

Lemma 1.1. If v is different from w, v does not occur free in W, and Y_2 is free for v in Y_1 then $[W/w][Y_2/v]Y_1 \equiv [[W/w]Y_2/v][W/w]Y_1$.

The proof is by induction on the structure of Y_1, the only non trivial case being when Y_1 is of the form λuX and v occurs free in Y_1. This means that u is different from v. If u is identical to w then w does not occur free in Y_2,since Y_2 is free for v in Y_1, so the relation follows. Finally if u is different from w then the substitutions can be reduced to X and the induction hypothesis can be applied.

Lemma 1.2 . Under the same assumptions of Lemma 1.1, if W is free for w in $[Y_2/v]Y_1$ then $[W/w]Y_2$ is free for v in $[W/w]Y_1$.

Assume the conclusion is false. Then w occurs free in Y_2 and W is not free for v in Y_1. But this contradicts that W is free for w in $[Y_2/v]Y_1$.

1.3 Reduction in BAL is an extension of the standard reduction in the lambda calculus. We define *redex* and the *contractum* of a redex by the following rules:

RX1 : If X is a term of the form λvY then X is a redex of type I. Any term of the form $\lambda u[u/v]Y$ where u is strictly free for v is a contractum of X.

RX2 : If X is of the form λvYZ then X is a redex of type II. If Z is free for v in Y then $[Z/v]Y$ is the contractum of X.

RX3 : If X is of the form $(X_1 \supset X_2,X_3)Y$ then X is a redex of type III and $(X_1 \supset X_2Y,X_3Y)$ is the contractum of X.

If X is a term containing a redex U as a subterm, and Y is the result of replacing U by some contractum, we say that X *reduces immediately to* Y and write X red_m Y.

We define the relation X *reduces to* Y as the reflexive and transitive closure of the relation X red_m Y. If X reduces to Y we write X red Y.

To prove that the relation reduction satisfies the Church-Rosser Property (CHRP) we shall follow the approach in [1]. First we define another relation red_s such that reduction is the reflexive and transitive closure of red_s. Then it is sufficient to prove the CHRP for the relation red_s.

1.4 We define the relation red_s inductive by a set of rules, each involving a set of premises, that in some cases may be empty. We describe the premises in advance for all rules and then proceed to describe the conclusion for each rule.

If X_1 red_s Y_1, ... , X_n red_s Y_n, $n \geq 0$, then:

Rule A: If X_0 is an atomic term then

$$X_0X_1\ldots X_n \text{ red}_s\ X_0Y_1\ldots Y_n$$

Rule B: If $n \geq 1$ and the variable u is strictly free for the variable v in Y_1 then

$$\lambda vX_1\ldots X_n \text{ red}_s\ \lambda u[u/v]Y_1\ldots Y_n$$

Rule C: If $n \geq 2$, u is strictly free for v in X_1, and

Y_2 is free for v in Y_1 then

$$\lambda u[u/v]X_1X_2\ldots X_n \text{ red}_s [Y_2/v]Y_1Y_3\ldots Y_n$$

Rule D: If $n \geq i \geq 3$ then

$$(X_1 \supset X_2,X_3)X_4\ldots X \text{ red}_s$$
$$(Y_1 \supset Y_2Y_4\ldots Y_i,Y_3Y_4\ldots Y_i)Y_{i+1}\ldots Y_n$$

Rule E: If $n \geq 1$ and $i \geq 0$ then

$$\{X_1\}_iX_2\ldots X_n \text{ red}_s \{Y_1\}_iY_2\ldots Y_n$$

It is clear that X red_s Y implies X red Y. To prove that red is the reflexive and transitive closure of red_s it is sufficient to show that X red_m Y implies X red_s Y.

Lemma 1.3 Assume X_1 red_s Y_1, X_2 red_s Y_2 and X_3 red_s Y_3. Then λvX_1 red_s λvY_1, $(X_1 \supset X_2,X_3)$ red_s $(Y_1 \supset Y_2,Y_3)$ and $\{X_1\}_i$ red_s $\{Y_1\}_i$.

This follows immediately from rules B,D and E.

Lemma 1.4 If X red_s Y and V red_s W then XV red_s YW.

If X red_s Y then some of the defining rules must apply. If we enlarge the set of premises of the rule by including V red_s W then by the same rule we get XV red_s YW.

Lemma 1.5 If X red_m Y then X red_s Y.

The proof is by induction on the construction of the term X. Note that in case X is a redex and Y is a contractum then X red_s Y follows using rules B,C, and D. Then proceed by induction using lemmas 1.3 and 1.4.

Lemma 1.6 If X red_s Y and v is a variable that occurs free in Y then v occurs free in X.

This is clear from the form of the defining rules.

Theorem 1.1 If X red_s Y, V red_s W, V is free for w in X and W is free for w in Y, then $[V/w]X$ red_s $[W/w]Y$.

The proof is by induction on the construction of X, considering cases according to the rule used to derive X red_s Y. In all cases we use the following notation. If the premises in the rule are: X_1 red_s Y_1, ... , X_n red_s Y_n, then we put $X_j' \equiv [V/w]X_j$, and $Y_j' \equiv [W/w]Y_j$, $j = 1,\dots,n$.

Rule B. Here $X \equiv \lambda v X_1 \dots X_n$, and $Y \equiv \lambda u[u/v]Y_1 \dots Y_n$. By the induction hypothesis we have X_j' red_s Y_j', $j = 2,\dots,n$. If $v \equiv w$ or $u \equiv w$ it is sufficient to take X_1 red_s Y_1 as a premise and apply rule B. If $v \not\equiv w$ and $u \not\equiv w$ then V is free for w in X_1 and W is free for w in Y_1 so X_1' red_s Y_1' follows. In order to apply rule B we have to make sure that u is strictly free for v in Y_1'. Clearly we may assume that w occurs free in Y_1. It follows that w occurs free in X_1, u does not occur free in W and v does not occur free in V or W. Hence

$$\begin{aligned}[v/u][u/v]Y_1' &\equiv [v/u][u/v][W/w]Y_1 \\ &\equiv [v/u][W/w][u/v]Y_1 \\ &\equiv [W/w][v/u][u/v]Y_1 \\ &\equiv Y_1'\end{aligned}$$

If we apply rule B we get the desired relation for

$$\begin{aligned}\lambda v X_1' &\equiv [V/w]\lambda v X_1 \\ \lambda u[u/v]Y_1' &\equiv \lambda u[u/v][W/w]Y_1 \\ &\equiv \lambda u[W/w][u/v]Y_1 \\ &\equiv [W/w]\lambda u[u/v]Y_1\end{aligned}$$

Rule C. Here $X \equiv \lambda u[u/v]X_1X_2\ldots X_n$ and $Y \equiv [Y_2/v]Y_1Y_3\ldots Y_n$. Note first that the variable v is used only to describe substitutions. We can use the induction hypothesis on X_1 to replace such variable. Hence we may assume that $v \not\equiv u$ and $v \not\equiv w$ and also that v does not occur free in V or W. It is clear that V is free for w in $X_1,\ldots,X_n$ and W is free for w in $Y_1,Y_3,\ldots,Y_n$. In case v does not occur free in Y_1 then Y_2 does not appear in Y, so we may replace Y_2 in such a way that W is free for w in Y_2. This means that in any case we have X_j' red_s Y_j', $j = 1,\ldots,n$. To apply rule C we need first that u is strictly free for v in X_1', and for this we may assume that w occurs free in X_1. It follows that $u \not\equiv w$ and u does not occur free in V, hence

$$\begin{aligned}[v/u][u/v]X_1' &\equiv [v/u][u/v][V/w]X_1\\ &\equiv [v/u][V/w][u/v]X_1\\ &\equiv [V/w][v/u][u/v]X_1\\ &\equiv X_1'\end{aligned}$$

We need also that Y_2' is free for v in Y_1' but this follows from Lemma 1.2. Now we may apply rule C, and this gives the desired relation. For first note that by Lemma 1.1 we have $[Y_2'/v]Y_1' \equiv [W/w][Y_2/v]Y_1$. Furthermore we have also $\lambda u[u/v]X_1' \equiv [V/w]\lambda u[u/v]X_1$ which is trivial in case $w \equiv u$ (so w does not occur free in X_1) and in case $w \not\equiv u$ it follows because v does not occur free in V.

The other rules are trivial. This completes the proof of Theorem 1.1.

Theorem 1.2 If X red_s Y and X red_s Z then there is a term U such that Y red_s U and Z red_s U.

The proof is by induction on the structure of X with cases arising from the rules used in both derivations. The premises of $X \text{ red}_s Y$ are denoted as $X_1 \text{ red}_s Y_1, \ldots, X_n \text{ red}_s Y_n$ and the premises of $X \text{ red}_s Z$ as $X_1 \text{ red}_s Z_1, \ldots, X_n \text{ red}_s Z_n$. By the induction hypothesis we may assume that for $j = 1,\ldots,n$ there is a term U_j such that $Y_j \text{ red}_s U_j$ and $Z_j \text{ red}_s U_j$. We shall assume also that these terms can be chosen in such a way that collision with free variables are avoided.

Rules A, D and E are trivial. Rules B and C may appear in three combinations: both rule B, both rule C, one rule B and the other rule C.

Both rule B. Here $X \equiv \lambda v X_1 \ldots X_n$ and $Y \equiv \lambda u[u/v]Y_1 \ldots Y_n$, and $Z \equiv \lambda u'[u'/v]Z_1 \ldots Z_n$. Note now that for a properly chosen variable w we have

$$[u/v]Y_1 \text{ red}_s [u/v]U_1 \text{ and } [w/u][u/v]U_1 \equiv [w/v]U_1$$
$$[u'/v]Z_1 \text{ red}_s [u'/v]U_1 \text{ and } [w/u'][u'/v]U_1 \equiv [w/v]U_1$$

so by rule B we get

$$Y \text{ red}_s \lambda w[w/v]U_1 \ldots U_n$$
$$Z \text{ red}_s \lambda w[w/v]U_1 \ldots U_n$$

Both rule C. Here we note that the variable v in the rule can be chosen the same in both derivations. It follows from Theorem 1.1 that we may choose $U \equiv [U_2/v]U_1U_3 \ldots U_n$.

Rules B and C. Here $X \equiv \lambda u[u/v]X_1X_2 \ldots X_n$, $Y \equiv [Y_2/v]Y_1Y_3 \ldots Y_n$, and $Z \equiv \lambda w[w/u]Z_1'Z_2 \ldots Z_n$. We have of course that $X_1 \text{ red}_s Y_1$ but only that $[u/v]X_1 \text{ red}_s Z_1'$. Since we can choose v to be strictly free for u in Z_1'

it follows that X_1 red_s $[v/u]Z_1' \equiv Z_1$ and now we apply the induction hypothesis so Y_1 red_s U_1 and Z_1 red_s U_1. But then Z_1' red_s $[u/v]U_1$ and it follows that we may take $U \equiv [U_2/v]U_1U_3\ldots U_n$.

As mentioned above from Theorem 1.2 it follows that the CHRP holds for the relation reduction. Hence if X red Y and X red Z there is U such that Y red U and Z red U.

1.5 A term containing no redex of type II or III is said to be in *normal form*. In general given a term X there is no term Y in normal form such that X red Y. And whenever exists, it is unique up to change of bound variables. In the next section we introduce an evaluation procedure that applies only to terms in normal form. To be able to extend this evaluation we follow [8] and associate with each term X a partial normal form denoted as X^p. The definition of X^p is given by the following induction rules:

P1: If $X \equiv X_0X_1\ldots X_n$, $n \geq 0$, where X_0 is atomic then $X^p \equiv X_0X_1^p\ldots X_n^p$.

P2: If $X \equiv \lambda vX_1$ then $X^p \equiv \lambda vX_1^p$

P3: If $X \equiv \lambda vX_1X_2\ldots X_n$, $n \geq 2$, then $X^p \equiv ([\Omega/v]X_1X_3\ldots X_n)^p$

P4: If $X \equiv (X_1 \supset X_2,X_3)X_4\ldots X_n$, $n \geq 3$, then $X^p \equiv (X_1^p \supset (X_2X_4\ldots X_n)^p,(X_3X_4\ldots X_n)^p)$

P5: If $X \equiv \{X_0\}_1X_1\ldots X_n$, $n \geq 0$, then $X^p \equiv \{X_0^p\}_1X_1^p\ldots X_n^p$

Lemma 1.7 If Y is atomic and Y is free for u in X then $([Y/u]X)^p \equiv [Y/u]X^p$.

The proof by induction on the structure of X is straightforward.

We are interested in determining the relation between terms X^p and Y^p whenever X red Y holds. For this purpose we introduce the relation X *is Ω-covered by* Y, where X and Y are terms in normal form. We denote this relation in the form X Ω-cov Y, and it is defined by the following rules:

CV1: If $Y, X_1, \ldots, X_n$, $n \geq 0$, are terms in normal form then $\Omega X_1 \ldots X_n$ Ω-cov Y.

CV2: If X_1 Ω-cov $Y_1, \ldots, X_n$ Ω-cov Y_n, $n \geq 0$, and X_0 is atomic, then $X_0 X_1 \ldots X_n$ Ω-cov $X_0 Y_1 \ldots Y_n$.

CV3: If X_1 Ω-cov Y_1, and u is strictly free for v in Y_1, then $\lambda v X_1$ Ω-cov $\lambda u[u/v]Y_1$.

CV4: If X_1 Ω-cov Y_1, X_2 Ω-cov Y_2, and X_3 Ω-cov Y_3, then $X_1 \supset X_2, X_3$ Ω-cov $Y_1 \supset Y_2, Y_3$.

CV5: If X_1 Ω-cov Y_1, ... , X_n Ω-cov Y_n, $n \geq 1$, then $\{X_1\}_1 X_2 \ldots X_n$ Ω-cov $\{Y_1\}_1 Y_2 \ldots Y_n$.

It is easy to show that if X is any term in normal form then X Ω-cov X holds.

The notation X Ω-cov* Y denotes the transitive closure of the relation X Ω-cov Y.

Lemma 1.8. Let X and Y be arbitrary terms such that Y is free for w in X. Then $([\Omega/w]X)^p$ Ω-cov $([Y/w]Y)^p$.

The proof is by induction on the structure of X. If $X \equiv X_0 X_1 \ldots X_n$ and X_0 is w, the conclusion follows by rule CV1. If X_0 is atomic different from w, it follows by the induction hypothesis.

If $X \equiv \lambda v X_1 X_2 \ldots X_n$, $n \geq 2$, take $Z \equiv [\Omega/v]X_1 \ldots X_3$. It is clear that $([\Omega/v]X)^p \equiv ([\Omega/v]Z)^p$. Using that Y is free for w in X we have $([Y/w]X)^p \equiv ([Y/w]Z)^p$. Hence the conclusion follows by the induction hypothesis on Z. The other cases are similar.

Theorem 1.3. If X red Y then X^p Ω-cov* Y.

From the definitions it follows that we need only to prove that if X red_s Y then X^p Ω-cov Y^p. The proof is by induction on the structure of X, with cases arising from the rules in the derivation of X red_s Y. For example, assume that rule C is used. In this case $X \equiv \lambda u[u/v]X_1 X_2 \ldots X_n$ and $Y \equiv [Y_2/v]Y_1 \ldots Y_n$. Note that the variable v is used only to describe substitutions, hence we may assume that v does not occur in the terms $X_2, \ldots, X_n, Y_2, \ldots, Y_n$. Now put $Z_1 \equiv X_1 X_3 \ldots X_n$, and $Z_2 \equiv Y_1 Y_3 \ldots Y_n$. It follows that $X^p \equiv ([\Omega/v]Z_1)^p$ and $Y \equiv [Y_2/v]Z_2$. By the induction hypothesis and Lemma 1.8 we have

$$X^p \ \Omega\text{-cov} \ ([\Omega/v]Z_2)^p \ \Omega\text{-cov} \ Y^p$$

1.6 We complete this section with a new definition. If X and Y are arbitrary terms, and there is a term U such that X red U and Y red U, we shall say that X is *convertible to* Y and write X conv Y.

From the CHRP it follows that the relation X conv Y is the reflexive, symmetric and transitive closure of the relation X red Y.

2. COMPUTABILITY WITH BAL

2.1 Let D be a partially ordered set under some relation $\sqsubseteq$. We say that D is a *domain* in case the following conditions are satisfied: i) D contains a least element $\bot$ (bottom), and a greatest element $\top$ (top), and they are different. ii) If M is a directed subset of D (i.e. if each finite subset of M has an upper bound in M) then the least upper bound of M in D exists and it is denoted $\sqcup M$. If D is affected by subscript, superscript or index then the symbols $\sqsubseteq$, $\bot$, $\top$, $\sqcup$ will be similarly affected.

A subset D' of the domain D is a *subdomain* of D if it is a domain under the restriction partial order and whenever M is a directed subset of D' then $\sqcup M = \sqcup' M$ (or equivalently $\sqcup M \in D'$).

If D is a partially ordered set then D^+ denotes the set obtained by adding two new elements $\bot$ and $\top$, and extending the ordering in such a way that they become the bottom and top of D^+. For example if A is any set we may consider A partially ordered by the identity relation. Then A^+ is a domain (actually a complete lattice), called the flat domain induced by A. In particular if A = {true,false} then we put Bool = A^+. Another application of this notation is ω^+ where ω is the set of nonnegative integers.

2.2 Let D and D' be domains. A function from D into D' is *continuous* in case $f(\sqcup M) = \sqcup' f(M)$ whenever M is a directed subset of D. The set of all continuous functions from D into D' is denoted by $D \to D'$.

The set $D \to D'$ becomes a domain if we introduce the partial order $f \sqsubseteq g$ if and only if $f(x) \sqsubseteq' g(x)$ for all $x \in D$. Then $\bot_{D\to D'} = \lambda y\varepsilon D.\bot_{D'}$ and $\top_{D\to D'} = \lambda y\varepsilon D.\top_{D'}$. Furthermore if F is a directed subset of $D \to D'$ then $\sqcup F = g \in D \to D'$ where for $x \in D$ we have $g(x) = \sqcup'\{ f(x) : f \in F\}$

The notation $D_1 \to D_2 \to D_3$ is an abbreviation of $D_1 \to (D_2 \to D_3)$. And in general we put

$$D_1 \to D_2 \to \ldots \to D_n = D_1 \to (D_2 \to \ldots \to D_n).$$

If $f \in D_1 \to \ldots \to D_n \to D'$ and $x_1 \in D_1, \ldots, x_n \in D_n$ we put $f(x_1,\ldots,x_n) = f(x_1)\ldots(x_n)$.

A *retraction* in the domain D is a function $f \in D \to D$ such that $f \circ f = f$. It follows that $f(D) = \mathrm{Fix}(f) = \{ x : f(x) = x\}$ is a subdomain of D. A retraction f such that $f(x) \sqsubseteq x$ for all $x \in D$ is called a *projection*.

An *embedding* of the domain D' into the domain D is a pair (g,h) where $g \in D' \to D$, $h \in D \to D'$ and $h(g(x)) = x$ for all $x \in D'$. It follows that $g \circ h$ is a retraction in D and D' is isomorphic to $\mathrm{Fix}(g \circ h)$.

Let $\{D_j : j \in J\}$ be a collection of domains indexed by the set J. We put $H = \{\langle j,x\rangle : j \in J \wedge x \in D_j\}$ and define the domain $D = \bigoplus_{j\varepsilon J} D_j = H^+$, where the ordering in H is the natural extension of the orderings in each domain D_j. We call D the direct sum of the indexed collection of domains. For each $j \in J$ there is a canonical embedding of D_j into D given by the pair (ℓ_j,q_j) where the function ℓ_j is defined by $\iota_j(x) = \langle j,x \rangle$, and q_j is defined by the following cases:

$$\begin{aligned} q_j(y) &= \bot_j \text{ if } y = \bot \\ &= x \text{ if } y = \langle j,x \rangle \\ &= \bot_j \text{ if } y = \langle i,x \rangle \text{ and } j \neq i \\ &= \top_j \text{ if } y = \top \end{aligned}$$

projections.

If D is some domain we define a function $\mathrm{Cond}_D \in \mathrm{Bool} \to D \to D \to D$ as follows:

$$\begin{aligned} \mathrm{Cond}_D(x,y,z) &= \bot_D \quad \text{if } x = \bot_{\mathrm{Bool}} \\ &= y \quad \text{if } x = \text{true} \\ &= z \quad \text{if } x = \text{false} \\ &= \top_D \quad \text{if } x = \top_{\mathrm{Bool}} \end{aligned}$$

The continuity of the function Cond_D can be easily verified. Note that $\mathrm{Cond}_{D \to D'}(x,y_1,y_2)(z) = \mathrm{Cond}_{D'}(x,y_1(z),y_2(z))$. If $f \in D \to D'$ is such that $f(\bot_D) = \bot_{D'}$ and $f(\top_D) = \top_{D'}$ then $f(\mathrm{Cond}_D(x,y,z)) = \mathrm{Cond}_{D'}(x,f(y),f(z))$.

2.3 A *basic structure* H is a collection $\{H_j : j \in \omega\}$ of domains indexed by ω such that $H_0 = \mathrm{Bool}$. The elements of H_j are called *ground elements* of type (j). More generally if $\alpha = (i1,\ldots,ik,j)$, $k \geq 0$, $i1,\ldots,ik,j \in \omega$ we say that α is a *ground type*, put $H_\alpha = H_{i1} \to \ldots\ldots \to H_{ik} \to H_j$ and say that the elements of H_α are ground operators of type α. Finally if $\beta = (\alpha 1,\ldots,\alpha n,j)$, $n \geq 0$ and $\alpha 1,\ldots,\alpha n$ are ground types, $j \in \omega$, we say that β is a *functional type*, put $H_\beta = H_{\alpha 1} \to \ldots \to H_{\alpha n} \to H_j$ and say that the elements of H_β are functionals of type β .

An *algorithmic system* is a triple S = (H,C,int) where H is a basic structure, C is a set of constants in BAL containing at least the symbols tt and ff, and int is a function such that for each constant $c \in C$ int(c) = some ground operator in H, int(tt) = true, int(ff) = false. If int(c) if a ground operator of type α we say that c is of type α in S.

An *assignment* in the system S is a function σ such that for any variable v in BAL $\sigma(v)$ = some ground operator in H. If $\sigma(v)$ is of type α we say that v is of type α in S under σ.

We fix now an algorithmic system S = (H,C,int). We shall consider only terms in which the constants are in C. We proceed to define a function $Ev_i^S(X)(\sigma)$ where $i \in \omega$,X is a term in normal form and σ is an assignment in S. The value of this function is some element of H_i. To simplify the notation we shall not write the superscript S. The evaluation function is defined by the following rules:

EV1: If $X \equiv X_0X_1\ldots X_n$, $n \geq 0$, where X_0 is a constant of type $\alpha = (i1,\ldots,ik,i)$ then

$$Ev_i(X)(\sigma) = int(X_0)(Ev_{i1}(X_1)(\sigma),\ldots,Ev_{ik}(X_k)(\sigma))$$

where in case n < 0 we put $X_{n+1} \equiv \ldots \equiv X_k \equiv \Omega$

EV2: If $X \equiv X_0X_1\ldots X_n$, $n \geq 0$, where X_0 is a variable of type $\alpha = (i1,\ldots,ik,i)$ under σ then

$$Ev_i(X)(\sigma) = \sigma(X_0)(Ev_{i1}(X_1)(\sigma),\ldots,Ev_{ik}(X_k)(\sigma))$$

where in case n < 0 we put $X_{n+1} \equiv \ldots \equiv X_k \equiv \Omega$

EV3: If $X \equiv X_0X_1\ldots X_n$, $n \geq 0$, where X_0 is atomic and neither rule EV1 nor rule EV2 applies then $Ev_i(X)(\sigma) = \perp_i$

EV4: If $X \equiv \lambda v X_1$ then $Ev_i(X)(\sigma) = Ev_i([\Omega/v]X_1)(\sigma)$

EV5: If $X \equiv X_1 \supset X_2, X_3$ then

$Ev_i(X)(\sigma) = Cond_{H_i}(Ev_0(X_1)(\sigma), Ev_i(X_2)(\sigma), Ev_i(X_3)(\sigma))$

EV6: If $X \equiv \{X_0\}_j X_1 \ldots X_n$, $n \geq 0$, and $i = j$ then $Ev_i(X)(\sigma) = Ev_i(X_0)(\sigma)$. If $i \neq j$ then $Ev_i(X)(\sigma) = \perp_i$

Note that from EV1 it follows that $Ev_0(ttX_1\ldots X_n)(\sigma) =$ true and $Ev_0(ffX_1\ldots X_n)(\sigma) =$ false. And from EV3 it follows that $Ev_i(\Omega X_1\ldots X_n)(\sigma) = \perp_i$.

Theorem 2.1. If X Ω-cov Y and σ is any assignment then $Ev_i(X)(\sigma) \subseteq_i Ev_i(Y)(\sigma)$.

Proof by induction on the structure of X with cases arising from the rules defining the covering relation. All cases are straightforward.

We proceed now to extend the evaluation function to arbitrary terms. Note that from the CHRP, Theorem 1.3 and Theorem 2.1 it follows that for any term X the collection $\{Ev_i(Y^p)(\sigma) : X \text{ red } Y\}$ is directed. We define then $Ev_i(X)(\sigma) = \cup_i\{Ev_i(Y^p)(\sigma) : X \text{ red } Y\}$

Let F be a functional of type $\beta = (\alpha 1, \ldots, \alpha n, i)$. We say that the closed term X *computes* F in S in case that for arbitrary $x_1, \ldots, x_n$ of the proper type the following relation holds:

$$F(x_1, \ldots, x_n) = Ev_i(Xv_1\ldots v_n)(\sigma)$$

where $\sigma(v_j) = x_j$, $j = 1,\ldots,n$. Finally we say that the functional F is *S-computable* in case there is a closed term X that computes F in S.

2.4 We want to prove some general properties of S-computable functionals, essentially closure under substitution and recursion. In principle this is possible using the above definitions, but this approach is involved and requires a great deal of syntactical analysis. Some examples will illustrate the general situation.

Let $Y \equiv \lambda v_1(MM)$ where $M \equiv \lambda v_2(v_1(v_2v_2))$. We want to determine the functional F of type $((\iota,\iota),\iota)$ computed by Y. It is easy to see that whenever Yv_1 red Y then Y is either of the form $\lambda v_1(v_1^n(MM))v_1$ or of the form $v_1^n(MM)$. It follows that Y^p is either $\Omega^{n+1}(\Omega\Omega)$ or $v_1^{n+1}(\Omega\Omega)$ so if we put $\sigma(v_1) = f$ of type (ι,ι) it follows that $Ev_1(Y^p)(\sigma) = f^n(\bot_1)$. This means that $F(f) =$ minimal fixed point of f.

Let consider now an example involving substitution. Assume $Ev_j(V)(\sigma) = e \in H_j$ and also that V is free for v in X. In this case we may expect $Ev_i([V/v]X)(\sigma) = Ev_i(X)(\sigma')$ where $\sigma'(v) = e$ and otherwise σ' is identical with σ. This is not true in general but it is true if we replace V by $\{V\}_j$, i.e. if we isolate V in X. Similar isolation techniques are necessary for more general substitutions.

The crucial problem seems to be that the evaluation of a term X is not directly determined by the structure of X. It is certainly determined by the structure of X, but indirectly, through a reduction procedure that generates other terms whose structure is in general difficult to predict. In this sense we may say that the evaluation described in this section is a reductional semantics for BAL. We would like to have a structural semantics in which the evaluation of a term is directly determined by its structure. The possibility of such a semantics was discovered by D. Scott (see [5]), and the equivalence of both semantics for the lambda calculus was proved by Wadsworth (see [8]).

3. STRUCTURAL SEMANTICS

3.1 A domain D such that there is an embedding of $D \to D$ into D is called a *reflexive domain.* The structural semantics is obtained by embedding the basic structure H in some reflexive domain D. We must impose a number of restrictions relating these embeddings.

A *reflexive embedding* of the structure H consists of a domain D, a pair (ψ,ϕ) embedding $D \to D$ into D, and for each $i \in \omega$ a pair (g_i,h_i) embedding H_i into D, such that the following conditions are satisfied for arbitrary i,j in ω:

RE1: If $i \neq j$ then $h_i \circ g_j = \lambda y \in H_j . \perp_i$

RE2: For each $x \in H_i$ $\phi(g_i(x)) = \lambda d \in D . g_i(x)$

RE3: For each $f \in D \to D$ $h_i(\psi(f)) = h_i(f(\perp))$

We shall assume some reflexive embedding E of the basic structure H, and prove some elementary consequences of the conditions RE1-RE3. Then we assume an algorithmic system S and define the structural semantics. The equivalence with the reductional semantics is proved assuming the reflexive embedding is minimal. A minimal reflexive embedding is constructed in the next section.

3.2 First it is convenient to generalize the function ϕ. We put $D^0 = D$ and $D^{n+1} = D \to D^n$. Then we define $\phi_n \in D \to D^n$ as follows: $\phi_0 = I_D$, $\phi_{n+1}(d) = \phi_n \circ \phi(d)$.

Note that $\phi(\perp) = \lambda d \in D . \perp$, and $\phi(\top) = \lambda d \in D . \top$. Hence the following relation holds for arbitrary elements d and d' of D and $x \in$ Bool:

$$\phi(\mathrm{Cond}_D(x,d,d')) = \mathrm{Cond}_{D\to D'}(x,\phi(d),\phi(d'))$$

Lemma 3.1. If $n \geq 0$ and $d_0, d_1, \ldots, d_{n+1}$ are elements of D then

$$\phi_{n+1}(d_0)(d_1, \ldots, d_{n+1}) = \phi(\phi_n(d_0)(d_1, \ldots, d_n))(d_{n+1})$$

The proof is by induction on n. The case $n = 0$ is trivial. Assume the relation holds for n. Then

$$\begin{aligned}\phi_{n+2}(d_0)(d_1, \ldots, d_{n+2}) &= \phi_{n+1}(\phi(d_0)(d_1))(d_2, \ldots, d_{n+2}) \\ &= \phi(\phi_n(\phi(d_0)(d_1))(d_2, \ldots, d_{n+1}))(d_{n+2}) \\ &= \phi(\phi_{n+1}(d_0)(d_1, \ldots, d_{n+1}))(d_{n+2})\end{aligned}$$

Lemma 3.2. If $x \in H_i$ and $d_1, \ldots, d_n$ are elements of D,then $\phi_n(g_i(x))(d_1, \ldots, d_n) = g_i(x)$.

This follows immediately from RE2 and the definition of ϕ_n.

3.3 The next step is to define embeddings for the domains of ground operators in the basic structure. From now on in this section the letter α will denote a ground type $(i1, \ldots, ik, j)$ where $k \geq 0$. In case $k > 0$ then α' will denote the ground type $(i2, \ldots, ik, j)$.

We define embeddings (g_α, h_α) of H_α into D by induction on k. If $k = 0$ we put $g_\alpha = g_j$ and $h_\alpha = h_j$. If $k > 0$ we define

$$g_\alpha(f) = \psi(g_{\alpha'} \circ f \circ h_{i1})$$
$$h_\alpha(d) = h_{\alpha'} \circ \phi(d) \circ g_{i1}$$

Lemma 3.3. Let f be a ground operator of type α. Then for arbitrary elements $d_1, \ldots, d_n$ of D, $n \geq 0$ the following relation holds:

$$\phi_n(g_\alpha(f))(d_1, \ldots, d_n) = g_\rho(f(h_{i1}(d_1), \ldots, h_{ik'}(d_{k'})))$$

where $k' = \min(n,k)$, $\rho = (j)$ in case $n \geq k$ and $\rho = (in+1, \ldots, ik, j)$ in case $k > n$.

Proof by induction on min(n,k). If $k = 0$ or $n = 0$ then both sides evaluate to $g_\alpha(f)$. If $n > 0$ and $k > 0$ then

$$\phi_n(g_\alpha(f)) = \phi_{n-1} \circ g_{\alpha'} \circ f \circ h_{11}$$

so by the induction hypothesis

$$\phi_{n-1}(g_{\alpha'}(f(h_{11}(d_1))))(d_2,\dots,d_n) =$$

$$g_\rho(f(h_{11}(d_1),\dots,h_{1k'}(d_{k'})))$$

where ρ and k' satisfy the conditions of the lemma.

Lemma 3.4. Let f be a ground operator of type α. Then $h_j(g_\alpha(f)) = f(\bot_{11},\dots,\bot_{1k})$. If $i \neq j$ then $h_i(g_\alpha(f)) = \bot_i$

If $k = 0$ this is trivial. If $k > 0$ then using RE3, $h_{11}(\bot) = \bot_{11}$ and the induction hypothesis on α' we have

$$\begin{aligned} h_j(g_\alpha(f)) &= h_j(\psi(g_{\alpha'} \circ f \circ h_{11})) \\ &= h_j(g_{\alpha'}(f(h_{11}(\bot)))) \\ &= f(\bot_{11},\dots,\bot_{1k}) \end{aligned}$$

If $i \neq j$ the argument is similar using RE1 in case $k = 0$.

Lemma 3.5. Let f be a ground operator of type α. Then for arbitrary elements $d_1,\dots,d_n$ of D, $n \geq 0$, the following relation holds:

$$h_j(\phi_n(g_\alpha(f))(d_1,\dots,d_n)) = f(h_{11}(d_1),\dots,h_{1k}(d_k))$$

where in case $k > n$ we put $d_{n+1} = \dots = d_k = \bot$

This follows using first Lemma 3.3 and in case $k > k'$ using Lemma 3.4.

3.4 We proceed now to define the structural semantics for BAL. Again we assume an algorithmic system $S = (H,C,\text{int})$ and also some reflexive embedding E of H. An assignment here is a function τ such that for any variable v, $\tau(v)$ is some element of D. The notation $[d/v]\tau$ denotes the assignment τ' where $\tau'(v) = d$ and otherwise τ' is identical with τ.

The semantics is given by a function $Va^S_E(X)(\tau)$ where X is any term and τ is any assignment. The function takes values in D. To simplify the notation we shall omit the subscript E and the superscript S. The definition is by induction according to the following rules:

VA1: If X is atomic then $Va(X)(\tau) = d$, where $d = g_\alpha(int(X))$ in case X is a constant of type α in S, $d = \tau(X)$ in case X is a variable, and $d = \bot$ in case $X \equiv \Omega$.

VA2: If $X \equiv YZ$ then $Va(X)(\tau) = \phi(Va(Y)(\tau))(Va(Z)(\tau))$

VA3: If $X \equiv \lambda vY$ then $Va(X)(\tau) = \psi(\lambda d \varepsilon D.Va(Y)([d/v]\tau))$

VA4: If $X \equiv X_1 \supset X_2, X_3$ then

$$Va(X)(\tau) = Cond_D(h_0(Va(X_1)(\tau)), Va(X_2)(\tau), Va(X_3)(\tau))$$

VA5: If $X \equiv \{Y\}_1$ then $Va(X)(\tau) = g_1(h_1(Va(Y)(\tau)))$

Technically the function Va is undefined if in rule VA3 the operator ψ applies to a non continuous function. But this situation does not arise and we can prove the following theorem.

Theorem 3.1. If X is any term and τ is any assignment then $Vx(X)(\tau)$ is defined. Furthermore if $\tau(v') = \sqcup M$ where M is a directed subset of D, then $Va(X)(\tau) = \sqcup\{Va(X)([d'/v']\tau): d' \varepsilon M\}$.

The proof by induction on the structure of X is straightforward. We consider only the case $X \equiv \lambda vY$. From the induction hypothesis it follows that $Va(X)(\tau)$ is defined. To prove the second part we may assume that the variable v' is different from v. For each $d' \in M$ we define a function $f_{d'}(d) = Va(Y)([d/v][d'/v']\tau)$. From the induction hypothesis it follows that $f_{d'} \in D \to D$. If d' and d" are elements of M such that $d' \subseteq d''$ then $f_{d'} \subseteq f_{d''}$ (this also follows from the induction hypothesis using $M' = \{d', d''\}$). Hence the collection $\{f_{d'} : d' \in M\}$ is directed in $D \to D$. Now using VA3 and the induction hypothesis we have

$$\begin{aligned} Va(X)(\tau) &= \psi(\lambda d\varepsilon D.\cup\{f_{d'}(d) : d' \in M\}) \\ &= \psi(\cup\{f_{d'} : d' \in M\}) \\ &= \cup\{\psi(f_{d'}) : d' \in M\} \\ &= \cup\{Va(X)([d'/v']\tau) : d' \in M\} \end{aligned}$$

The next two lemmas are standard basic results in structural semantics. Proof are omitted.

Lemma 3.6. Let τ and τ' be assignments such that $\tau(v) \subseteq \tau'(v)$ whenever v occurs free in X. Then $Va(X)(\tau) \subseteq Va(X)(\tau')$.

Lemma 3.7. If V is free for v' in X and $Va(V)(\tau) = d$, then $Va([V/v']X)(\tau) = Va(X)([d/v']\tau)$.

3.5 We are now in position to study the relation between reductional and structural semantics. We shall see that for terms in normal forms they are equivalent. For terms in general the situation is not clear. Recall the convention of 3.3 relative the ground type α.

Lemma 3.8. Let $X \equiv X_0X_1\ldots X_n$, $n \geq 0$, and

$Va(X_0)(\tau) = g_\alpha(f)$ for some ground operator f of type α. For $t = 1,\ldots,k$ let $d_t = Va(X_t)(\tau)$ where in case $k > n$ we put $X_{n+1} \equiv \ldots \equiv X_n \equiv \Omega$. Then $h_j(Va(X)(\tau)) = f(h_{i1}(d\ ),\ldots,h_{ik}(d_k))$ .Furthermore if $i \neq j$ then $h_i(Va(X)(\tau)) = \bot_i$.

First note that from Lemma 3.1 it follows that

$$Va(X)(\tau) = \phi_n(Va(X_0)(\tau))(Va(X_1)(\tau),\ldots,Va(X_n)(\tau))$$

Hence using Lemma 3.5 we get

$$h_j(Va(X)(\tau)) = f(h_{i1}(d_1),\ldots,h_{ik}(d_k))$$

If $i \neq j$ the $h_i(Va(X)(\tau)) = \bot_i$ follows from the second part of Lemma 3.4.

Theorem 3.2. Let X be a term in normal form, σ some assignment in S and τ an assignment such that whenever $\sigma(v) = f$ and f is a ground operator of type α then $\tau(v) = g_\alpha(\sigma(v))$. Then $Ev_i(X)(\sigma) = h_i(Va(X)(\tau))$.

The proof is by induction on the structure of X. Assume $X \equiv X_0X_1\ldots X_n$ where X_0 is atomic. In case $X_0 \equiv \Omega$ then both sides evaluate to $\bot_i$. Otherwise we use Lemma 3.8.

If $X \equiv \lambda vY$ then we have

$$\begin{aligned} Ev_i(X)(\sigma) &= Ev_i([\Omega/v]Y)(\sigma) \\ &= h_i(Va([\Omega/v]Y)(\tau)) \\ &= h_i(Va(Y)([\bot/v]\tau)) \quad \text{by Lemma 3.7} \\ &= h_i(Va(X)(\tau)) \text{ by RE3} \end{aligned}$$

The other cases follow easily from the definitions and the induction hypothesis.

If X and Y are terms such that for any assignment τ

we have $Va(X)(\tau) \subset Va(Y)(\tau)$ then we write $X \sqsubseteq Y$. If $X \sqsubseteq Y$ and $Y \sqsubseteq X$ hold we write $X \simeq Y$.

Lemma 3.9. Assume $X_1 \sqsubseteq Y_1$, $X_2 \sqsubseteq Y_2$ and $X_3 \sqsubseteq Y_3$. Then:

i) $X_1X_2 \sqsubseteq Y_1Y_2$

ii) $\lambda vX_1 \sqsubseteq \lambda vY_1$

iii) $X_1 \supset X_2,X_3 \sqsubseteq Y_1 \supset Y_2,Y_3$

iv) $\{X_1\}_j \sqsubseteq \{Y_1\}_j$

These relations follow immediately from the definitions and the monotonicity of the operations involved.

Lemma 3.10. If X is a redex and V is a contractum of X then $X \simeq V$.

Assume $X \equiv \lambda vX_1$ and $V \equiv \lambda u[u/v]X_1$ where u is strictly free for v in X_1. This means $X_1 \equiv [v/u][u/v]X_1$ and v is free for u in $[u/v]X_1$. Hence from Lemma 3.7 it follows that for any assignment τ and element d of D we have

$$Va(X_1)([d/v]\tau) = Va([u/v]X_1)([d/u]\tau)$$

and this implies $X \simeq V$.

For the other cases use Lemma 3.7 and the properties of the operator Cond discussed in 2.2.

Theorem 3.3. If X conv Y then $X \simeq Y$.

Proof by induction on the structure of X, using Lemmas 3.9 and 3.10.

Theorem 3.4. If X Ω-cov Y then $X \sqsubseteq Y$.

Proof by induction on the structure of X with cases arising from the rules defining the covering relation. All cases are straightforward.

Lemma 3.11. If X is any term then $X^p \sqsubseteq X$.

Proof by induction on the structure of X. All cases are easy. For example if $X \equiv \lambda v X_1 \ldots X_n$, $n \geq 2$, then by the induction hypothesis

$$X^p \equiv ([\Omega/v]X_1X_3\ldots X_n)^p \sqsubseteq [\Omega/v]X_1X_3\ldots X_n$$

and on the other hand using Lemmas 3.6 and 3.7

$$[\Omega/v]X_1X_3\ldots X_n \sqsubseteq X$$

From Theorem 3.4 it follows that for any term X and assignment τ the collection $\{Va(Y^p)(\tau) : X \text{ red } Y\}$ is directed. We shall say that the embedding E is *minimal* if the relation $Va(X)(\tau) = \sqcup\{Va(Y^p)(\tau) : X \text{ red } Y\}$ holds for arbitrary term X and assignment τ.

Theorem 3.5. Assume the embedding E is minimal, X is any term, σ is some assignment in S, and τ is an assignment such that whenever $\sigma(v) = f$ and f is a ground operator of type α, then $\tau(v) = g_\alpha(\sigma(v))$. Then $Ev_1(X)(\sigma) = h_1(Va(X)(\tau))$.

$$\begin{aligned} Ev_1(X)(\sigma) &= \sqcup\{Ev_1(Y^p)(\sigma) : X \text{ red } Y\} \\ &= \sqcup\{h_1(Va(Y^p)(\tau)) : X \text{ red } Y\} \\ &= h_1(\sqcup\{Va(Y^p)(\tau) : X \text{ red } Y\}) \\ &= h_1(Va(X)(\tau)) \end{aligned}$$

4. A MINIMAL EMBEDDING

4.1 To construct a minimal embedding we shall use the following procedure. We introduce a number of conditions and prove that an embedding in which they are satisfied is minimal. Then we construct an embedding in which those conditions hold.

Let E be an embedding with domain D where (ψ,ϕ) is the embedding of $D \to D$ into D. We shall assume there is a sequence P_0, P_1, ..., P_k, ... of projections in D such that the following conditions are satisfied:

RD1: For each $k \geq 0$, $P_k \subset P_{k+1}$, $P_k(\top) = \top$, and furthermore $\cup\{P_k : k \geq 0\} = I_{D\to D}$

RD2: For $k \geq 0$ and $d \in D$, $\phi(P_{k+1}(d)) \subset P_k \circ \phi(d) \circ P_k$

RD3: For $d \in D$, $\phi(P_0(d)) \subset P_0 \circ \phi(d) \circ \lambda d\epsilon D.\bot$

Now we extend the language BAL by introducing new symbols Π_k for each $k \geq 0$. We extend also the definition of terms by including a new clause:

T6: If X is a term then for each $k \geq 0$, $\Pi_k[X]$ is a term.

A *primitive* term is a term in which there is no occurrence of the symbols Π_k. The rank of a term X is the greatest k such that Π_k occurs in X, and it is 0 in case it is a primitive term. We denote by r(X) the rank of X.

The notion of redex and normal form is taken exactly as in 1.5. Note that a term of the form $\Pi_k[\lambda vY]Z$ may be in normal form.

If Y is a term we can eliminate from Y all occurrences of symbols Π_k, simply by replacing every part $\Pi_k[Z]$ by Z.

If Y is in normal form and X is the primitive term obtained by this elimination procedure, we shall say that Y is a *restricted form* of X. The set of all restricted forms of X is denoted by R(X). Note that if X is a primitive term in normal form then $X \in R(X)$.

The valuation procedure of the preceding section can be extended to non primitive terms by including a new rule:

VA6: If $X \equiv \Pi_k[Y]$ then $Va(X)(\tau) = P_k(Va(Y)(\tau))$

Since the functions P_k are continuous it follows that the general properties of the valuation function are still valid. We shall continue to use the notation of the preceding section.

By the assumption RD1 we know that $P_k \subset P_{k+1} \subset I$ If Y is some term, and Y' is obtained by eliminating from Y some symbols Π_k, or by replacing some symbols Π_k by Π_m where $k < m$, it follows that $Y \sqsubseteq Y'$. If Y is a restricted form of the term X, and Y' is in normal form then Y' is also a restricted form of X.

Note that if X is a primitive term the set $\{Va(X')(\tau) : X' \in R(X)\}$ is directed. For given two elements X' and X" of R(X) we may apply the elimination and replacement procedure explained above to obtain $Y \in R(X)$ such that $X' \sqsubseteq Y$ and $X'' \sqsubseteq Y$.

Theorem 4.1. If X is a primitive term then for every assignment τ:

$$Va(X)(\tau) = \sqcup\{Va(X')(\tau) : X' \in R(X)\}$$

The proof is by induction on the structure of X. The case X is atomic is trivial since then $X \in R(X)$. Assume $X \equiv YZ$. Note that if $Y' \in R(Y)$ and $Z' \in R(Z)$ then for any k, $\Pi_k[Y']Z' \in R(X)$. Hence we have

$$\begin{aligned} Va(X)(\tau) &= \phi(Va(Y)(\tau))(Va(Z)(\tau)) \\ &= \phi(\cup\{P_k(Va(Y)(\tau)):k\geq 0\})(Va(Z)(\tau)) \\ &= \cup\{\phi(P_k(Va(Y')(\tau)))(Va(Z')(\tau)):k\geq 0, Y'\in R(Y), Z'\in R(Z) \\ &= \cup\{Va(\Pi_k[Y']Z' : k\geq 0, Y'\in R(Y), Z'\in R(Z)\} \\ &= \cup\{Va(X')(\tau) : X' \in R(X)\} \end{aligned}$$

Now consider $X \equiv \lambda vY$. Here if $Y' \in R(Y)$ then $\lambda vY' \in R(X)$. We define $f_Y(d) = Va(Y)([d/v]\tau)$. And for $Y' \in R(Y)$ we define $f_{Y'}(d) = Va(Y')([d/v]\tau)$. From the induction hypothesis it follows that $f_Y = \cup\{f_{Y'} : Y' \in R(Y)\}$ Hence

$$\begin{aligned} Va(X)(\tau) &= \psi(f_Y) \\ &= \cup\{\psi(f_{Y'}) : Y' \in R(Y)\} \\ &= \cup\{Va(\lambda vY')(\tau) : Y' \in R(Y)\} \\ &= \cup\{Va(X')(\tau) : X' \in R(X)\} \end{aligned}$$

The other cases follow easily from the induction hypothesis and the continuity of the operations involved.

4.2 The proof of minimality is given via two lemmas that will be proved next. Note that from condition RD2 and Lemma 3.7 the following relation holds, provided that Y is free for v in X:

$$\Pi_{k+}[\lambda vX]Y \sqsubseteq \Pi_k[[\Pi_k[Y]/v]X]$$

Note also that from RD3 it follows that

$$\Pi_0[\lambda vX]Y \sqsubseteq \Pi_0[[\Omega/v]X]$$

Lemma 4.1. Let X be a primitive term and $X' \epsilon R(X)$ where $r(X) = k + 1$. Then there is a primitive term Y and $Y' \epsilon R(Y)$ such that X red Y, $r(Y') = k$, and $X' \sqsubset Y'$.

The proof is by induction on the structure of X. If $X \equiv X_0X_1...X_n$ where X_0 is atomic, we may assume that $X' \equiv X_0X_1'...X_n'$, where $X_i' \epsilon R(X_i)$, $i = 1,...,n$. So the result follows from the induction hypothesis.

Let $X \equiv \lambda vX_1...X_n$. If $n = 1$ we may assume that $X' \equiv \lambda vX_1'$, so the result follows from the induction hypothesis. If $n > 1$, then we may assume that $X' \equiv \Pi_{k+1}[\lambda vX_1']X_2'...X_n'$. By the induction hypothesis there are terms Y_i and $Y_i' \epsilon R(Y_i)$ such that X_i red Y_i, $r(Y_i') = k$, and $X_i' \sqsubset Y_i'$, $i = 1,....,n$. Clearly we can take Y_1 such that Y_2 is free for v in Y_1. Hence we take $Y \equiv [Y_2/v]Y_1Y_3...Y_n$ and $Y' \equiv \Pi_k[[\Pi_k[Y_2']/v]Y_1']Y_3'...Y_n'$. Clearly X red Y, $Y' \epsilon R(Y)$, $r(Y') = k$ and $X' \sqsubset Y'$ holds by RD2.

Assume $X \equiv (X_1 \supset X_2,X_3)X_4...X_n$. If $n = 3$ we may take $X' \equiv X_1' \supset X_2',X_3'$, and the result follows easily from the induction hypothesis. If $n > 3$ then we may assume that $X' \equiv \Pi_{k+1}[X_1' \supset X_2',X_3']X_4'...X_n'$. By the induction hypothesis we can find primitive terms Y_1, Z_1 and Z_2 such that X_1 red Y_1, $X_2X_4...X_n$ red Z_1, $X_3X_4...X_n$ red Z_2, and also terms Y_1', Z_1' and Z_2' such that $Y_1' \epsilon R(Y_1)$, $Z_1' \epsilon R(Z_1)$, $Z_2' \epsilon R(Z_2)$, $r(Y_1') = r(Z_1') = r(Z_2') = k$ and furthermore $X_1' \sqsubset Y_1'$, $\Pi_{k+1}[X_2']X_4'...X_n' \sqsubset Z_1'$, $\Pi_{k+1}[X_3']X_4'...X_n' \sqsubset Z_2'$. Hence we take $Y \equiv Y_1 \supset Z_1, Z_2$, and $Y' \equiv Y_1' \supset Z_1', Z_2'$. Now since $P_{k+1}(\bot) = \bot$, and $P_{k+1}(\top) = \top$, it follows that

$$X' \simeq X_1' \supset \Pi_{k+1}[X_2']X_4' \ldots X_n',\ \Pi_{k+1}[X_3']X_4' \ldots X_n' \sqsubseteq Y'$$

and clearly $r(Y') = k$.

The case $X \equiv \{X_0\}_j X_1 \ldots X_n$ follows easily from the induction hypothesis.

Lemma 4.2. If X is a primitive term, $X' \in R(X)$ and $r(X') = 0$, then $X' \sqsubseteq X^p$.

Proof by induction on the structure of X. All cases are easy, and we consider only $X \equiv \lambda v X_1 \ldots X_n$, $n \geq 2$. Here we may assume that $X' \equiv \Pi_0[\lambda v X_1']X_2' \ldots X_n'$. Hence using RD3 we have

$$\begin{aligned} X' &\sqsubseteq \Pi_0[[\Omega/v]X_1']X_3' \ldots X_n' \\ &\sqsubseteq ([\Omega/v]X_1 X_3 \ldots X_n)^p \quad \text{by ind. hyp.} \\ &\sqsubseteq X^p \end{aligned}$$

Theorem 4.2. If E is an embedding satisfying conditions RD1-RD3, then E is minimal.

Let X be some term and τ some assignment. From Theorem 4.1 it follows that we have to show only that for any $X' \in R(X)$ there is Y such that X red Y and $Va(X')(\tau) \subset Va(Y^p)(\tau)$. But this follows from Lemmas 4.1 and 4.2.

4.3 To construct a minimal embedding of a given basic structure H we follows the procedure of [5]. We define a sequence of domains D_0, D_1, ... such that D_k is a projection of D_{k+1}, and take the inverse limit of the sequence. We put $D_0 = \bigoplus_{i \epsilon \omega} H_i$, and $D_{k+1} = D_k \to D_k$. The embeddings of D_k into D_{k+1} are given by pairs (i_k, j_k) where $i_0(x) = \lambda y \epsilon D_0 . x$, $j_0(f) = f(\bot_0)$, $i_{k+1}(f) = i_k \circ f \circ j_k$, and $j_{k+1}(f) = j_k \circ f \circ i_k$. It is easy to check that all these functions are continuous, $j_k \circ i_k = I_k$, and $i_k \circ j_k \sqsubset_{k+1} I_{k+1}$. Note that $j_k(\top_{k+1}) = \top_k$, $j_k(\bot_{k+1}) = \bot_k$, $i_k(\top_k) = \top_{k+1}$ and $i_k(\bot_k) = \bot_{k+1}$

The inverse limit of this sequence is the set D of all functions d defined on ω such that for any $k \epsilon \omega$, $d(k) \epsilon D_k$ and $d(k) = j_k(d(k+1))$. If d and d' are elements of D we define $d \sqsubset d'$ if and only if $d(k) \sqsubset_k d'(k)$ for all $k \epsilon \omega$. Then D with this partial order is also a domain. For we may define $\bot(k) = \bot_k$, $\top(k) = \top_k$, so $\bot$ and $\top$ are least and greatest element of D. Furthermore if M is a directed subset of D then for each $k \epsilon \omega$, the set $\{d(k) : d \epsilon M\}$ is directed in D_k. So we may define $d'(k) = \sqcup_k\{d(k) : d \epsilon N\}$, by the continuity of the functions j_k it follows that $d' \epsilon D$ and actually $d' = \sqcup M$.

Next we define for each k an imbedding $(i_{k\infty}, j_{\infty k})$ of D_k into D. We put $j_{\infty k}(d) = d(k)$. To define $i_{k\infty}$ we use recursion. It is sufficient to define $i_{k\infty}(x)(m)$ for $x \epsilon D_k$ and $m \geq k$. We put $i_{k\infty}(x)(k) = x$ and $i_{k\infty}(x)(m+1) = i_m(i_{k\infty}(x)(m))$. It is easy to show that these functions are continuous, $j_{\infty k} \circ i_{k\infty} = I_k$, and $i_{k\infty} \circ j_{\infty k} \sqsubset I$. Note that $i_{k\infty}(\top_k) = \top$.and $j_{\infty k}(\top) = \top_k$

Lemma 4.3. Assume $d \in D$ and $m \geq k$. Then

i) $j_{\infty k} = j_k \circ j_{\infty k+1}$

ii) $i_{k\infty} = i_{k+1\infty} \circ i_k$

iii) $j_{\infty k} \circ i_{m\infty} \circ d(m+1) \circ j_{\infty m} \circ i_{k\infty} = d(k+1)$

Property i) follows immediately from the condition $j_k(d(k+1)) = d(k)$. To prove ii) we use induction to show that for $m \geq k+1$ and $x \in D_k$, $i_{k\infty}(x)(m) = i_{k+1\infty}(i_k(x))(m)$. Finally iii) is proved by induction on $m - k$. The case $m = k$ is trivial. For $m > k$ use i) and ii), the induction hypothesis and note the relation $j_k \circ d(k+2) \circ i_k = j_{k+1}(d(k+2)) = d(k+1)$.

Now we define projections P_k in D, $k \geq 0$. We put $P_k = i_{k\infty} \circ j_{\infty k}$. Note that $P_k \circ P_k = I$, and $P_k(\tau) = \tau$. Note also that $P_k = i_{k+1\infty} \circ i_k \circ j_k \circ j_{\infty k+1} \subset P_{k+1}$.

Lemma 4.4. Assume $m \geq k$. Then:

i) $j_{\infty k} \circ P_m = j_{\infty k}$

ii) $P_k \circ P_m = P_k$

iii) $P_m \circ i_{k\infty} = i_{k\infty}$

iv) $P_m \circ P_k = P_k$

v) $i_{m\infty} \circ P_{k+1}(d)(m+1) \circ j_{\infty m} = i_{k\infty} \circ d(k+1) \circ j_{\infty k}$

Both i) and iii) are proved by induction on $m - k$, using Lemma 4.3. Then ii) follows from i) and iv) follows from iii). The proof of v) uses also induction on $m - n$. The case $m = k$ is trivial. If $m > k$ use $P_{k+1}(d) = P_{k+2}(P_{k+1}(d))$, the induction hypothesis, and note that $P_{k+1}(d)(k+2) = i_k \circ d(k+1) \circ j_k$.

Lemma 4.5. $\cup\{P_m : m \geq 0\} = I_{D\to D}$

We put $d' = \cup\{P_m(d) : m \geq 0\}$. Then for any k

$$d'(k) = \cup_k\{P_m(d)(k) : m \geq k\}$$
$$= d(k) \text{ by Lemma 4.4 i)}$$

hence $d = d'$.

Corollary. If $f \in D \to D$, then $\cup\{P_m \circ f \circ P_m : m \geq 0\} = f$

Now we proceed to define the embedding (ψ,ϕ) of $D \to D$ into D. We put $\psi(f)(k+1) = j_{\infty k} \circ f \circ i_{k\infty}$, for $f \in D \to D$ and $k \geq 0$. And $\phi(d) = \cup\{i_{m\infty} \circ d(m+1) \circ j_{\infty m} : m \geq 0\}$ for $d \in D$. Note that ϕ is well defined for

$$\begin{aligned} i_{m\infty} \circ d(m+1) \circ j_{\infty m} &= i_{m+1\infty} \circ i_m \circ d(m+1) \circ j_m \circ j_{\infty m+1} \\ &= i_{m+1\infty} \circ i_{m+1}(j_{m+1}(d(m+2))) \circ j_{\infty m+1} \\ &\subset i_{m+1\infty} \circ d(m+2) \circ j_{\infty m+1} \end{aligned}$$

Note that $\phi(\psi(f)) = f$ follows immediately from Lemma 4.5. But we also have $\psi(\phi(d)) = d$ since $\psi(\phi(d))(k+1) = d(k+1)$ by Lemma 4.3 iii).

Theorem 4.3. The following relations hold for arbitrary $d \in D$ and $k \geq 0$ and $f \in D \to D$:

i) $\phi(P_{k+1}(d)) = i_{k\infty} \circ d(k+1) \circ j_{\infty k}$

ii) $P_k \circ \phi(d) \circ P_k = i_{k\infty} \circ d(k+1) \circ j_{\infty k}$

iii) $\phi(P_0(d)) = \lambda d' \in D.P_0(d)$

iv) $P_0(\psi(f)) = P_0(f(\bot))$

v) $\phi(P_0(d)) = P_0 \circ \phi(d) \circ \lambda d' \in D.\bot$

Property i) follows using the definition of ϕ and Lemma 4.4 v). Property ii) also follows using the definition of ϕ and Lemma 4.3 iii). To prove iii) we compute using part i) and the definition of i_0

$$\begin{aligned}\phi(P_0(d)) &= \phi(P_1(P_0(d))) \\ &= i_{0\infty} \circ P_0(d)(\bot) \circ j_{\infty 0} \\ &= i_{0\infty} \circ i_0(d(0)) \circ j_{\infty 0} \\ &= \lambda d' \varepsilon D. P_0(d)\end{aligned}$$

To prove iv) we compute using part iii) and the fact that P_0 is a projection

$$\begin{array}{ll} P_0(\psi(f)) \subset \psi(f) & P_0(f(\bot)) \subset f(\bot) \\ \phi(P_0(\psi(f))) \subset f & \phi(P_0(f(\bot))) \subset f \\ P_0(\psi(f)) \subset f(\bot) & P_0(f(\bot)) \subset \psi(f) \\ P_0(\psi(f)) \subset P_0(f(\bot)) & P_0(f(\bot)) \subset P_0(\psi(f)) \end{array}$$

To prove v) we use iv) with $f = \phi(d)$, hence

$$P_0(d) = P_0(\phi(d)(\bot))$$

so v) follows from iii).

Condition RD1 has been proved in Lemma 4.5. And conditions RD2 and RD3 follow from Theorem 4.3 i),ii) and v). To complete the proof we must define the embedding of H into D. Let recall that for each $i \varepsilon \omega$, there is a canonical embedding (ℓ_i, q_i) of H_i into D_0. Clearly if $i \neq j$ then $q_i(\ell_j(x)) = \bot_i$. Now we define $g_i = i_{0\infty} \circ \ell_i$ and $h_i = q_i \circ j_{\infty 0}$. It follows that $h_i \circ g_i = q_i \circ \ell_i = I_i$. Furthermore, noting that $P_0 \circ i_{0\infty} = i_{0\infty}$ and $j_{\infty 0} \circ P_0 = j_{\infty 0}$, and using Theorem 4.3 iii) and iv), we get RE2 and RE3.

5. DEFINABILITY

5.1 In this section we return to the S-computable functionals, and prove some closure properties. Although these results can be proved directly from the definitions, we rather use the minimal embedding. This makes possible a generalization which will be described in terms of definability.

We assume some algorithmic system S = (H,C,int) and a minimal embedding E of H. The domain of the embedding is D and (ψ,ϕ) is the embedding of $D \to D$ into D. We shall assume also that $\psi \circ \phi = I_D$, so we have an isomorphism of D and $D \to D$. This is not strictly necessary but it simplifies some of the proofs. If X is a closed term then $Va(X)(\tau)$ is actually independent of τ. So in this case we shall write Va(X).

If d_1 and d_2 are elements of D then the application operation in D is denoted $(d_1 d_2) = \phi(d_1)(d_2)$, so this is the counterpart of operation application in BAL. We follow the usual rules to eliminate parenthesis, namely association to the left. Note that $\lambda d' \varepsilon D.dd' = \phi(d)$, hence $\psi(\lambda d' \varepsilon D.dd') = d$. Furthermore if $d_1 d' = d_2 d'$ for all $d' \varepsilon D$ then $d_1 = d_2$.

We denote by Q_0 the subset of D consisting of all elements d such that for some closed term X, Va(X) = d. A basis in D is a subset Q of D such that Q_0 is a subset of Q, and Q is closed under application. It follows that Q_0 is a basis, the minimal basis contained in any other basis. A similar notion was defined in [4] where we required also that Q be countable.

There are some convenient rules to define elements in a given basis Q. For example there is $S \in Q_0$ such that for arbitrary d_1, d_2 and d_3 in D, $Sd_1d_2d_3 = d_1d_3(d_2d_3)$. Also there is $K \in Q_0$ such that $Kd_1d_2 = d_1$. We mention also elements B and B' such that $Bd_1d_2d_3 = d_1(d_2d_3)$, and $B'd_1d_2d_3d_4 = d_1(d_2(d_3d_4))$.

In general if M is an applicative expression containing variables from the list $d_1,\ldots,d_n$, constant symbols denoting elements of a basis Q, and the only operation in M is application, then there is a (unique) element $d \in Q$ such that $dd_1\ldots d_n = M$ holds for all values in D of the variables $d_1,\ldots,d_n$.

We recall the convention that the letter α always denote a ground type of the form $(i1,\ldots,ik,j)$, $k \geq 0$. The letter β will denote a functional type of the form $(\alpha 1,\ldots,\alpha n,i)$, $n \geq 0$.

Lemma 5.1. There is an element $r_\alpha \in Q_0$ such that for any $d \in D$, $r_\alpha d = g_\alpha(h_\alpha(d))$.

If $k = 0$ take $r_\alpha = Va(\lambda v_1\{v_1\}_j)$. If $k > 0$ note that $g_\alpha(h_\alpha(d)) = \psi(g_{\alpha'} \circ h_{\alpha'} \circ \phi(d) \circ g_{i1} \circ h_{i1})$ so take r_α such that $r_\alpha d = B'r_{\alpha'}dr_{i1}$.

5.2 Let F be a functional of type β. We say tha $d \in D$ *defines* F in case that the relation

$$F(x_1,\ldots,x_n) = h_i(dd_1\ldots d_n)$$

holds whenever $g_{\alpha j}(x_j) = d_j$, $j = 1,\ldots,n$. If Q is a basis and $d \in Q$ defines the functional F we say that F is *Q-definable*.

Theorem 5.1. A functional F is S-computable if and only if it is Q_0-definable.

This follows immediately from Theorem 3.5.

In order to study definability over some basis Q it is convenient to define embeddings for the domains H_β where β is a functional type. This is done essentially as for the ground types, and in fact the definitions agree in case β is actually a ground type.

The embedding of H_β is given again by a pair (g_β, h_β). If $n = 0$ we put $g_\beta = g_i$ and $h_\beta = h_i$. If $n > 0$ we put $g_\beta(F) = \psi(g_{\beta'} \circ F \circ h_{\alpha 1})$ and $h_\beta(d) = h_{\beta'} \circ \phi(d) \circ g_{\alpha 1}$, where $\beta' = (\alpha 2, \ldots, \alpha n, i)$.

Theorem 5.2. Let F be a functional of type β, and $g_\beta(F) = d$. If $n \geq m \geq 0$ and $\beta m = (\alpha m{+}1, \ldots, \alpha n, i)$ then the following relations hold for arbitrary $d_1, \ldots, d_m$:

i) $g_{\beta m}(F(h_{\alpha 1}(d_1), \ldots, h_{\alpha m}(d_m))) = dd_1 \ldots d_m$

ii) $d(r_{\alpha 1}d_1) \ldots (r_{\alpha m}d_m) = dd_1 \ldots d_m$

iii) $r_{\beta m}(dd_1 \ldots d_m) = dd_1 \ldots d_m$

To prove i) note that $g_{\beta 1}(F(h_{\alpha 1}(d_1))) = dd_1$, hence $g_{\beta 2}(F(h_{\alpha 1}(d_1), h_{\alpha 2}(d_2))) = dd_1d_2$, and so on. Now ii) follows from i) since $h_{\alpha j}(r_{\alpha j}d_j) = h_{\alpha j}(d_j)$, $j = 1, \ldots, m$. And iii) also follows from i) noting that $r_{\beta m}(g_{\beta m}(x)) = g_{\beta m}(x)$.

Theorem 5.3. Let Q be a basis and F a functional of type β. Then F is Q-definable if and only if $g_\beta(F) \in Q$.

If $g_\beta(F) = d \in Q$ then it is clear from Theorem 5.2 i), with $m = n$, that d defines F. On the other hand if $d' \in Q$ defines F take d such that $dd_1 \ldots d_n = r_1(d'(r_{\alpha 1}d_1)\ldots(r_{\alpha n}d_n))$. From this we get

$$F(h_{\alpha 1}(d_1),\ldots,h_{\alpha n}(d_n)) = h_1(d'(r_{\alpha 1}d_1)\ldots(r_{\alpha n}d_n))$$
$$g_1(F(h_{\alpha 1}(d_1),\ldots,h_{\alpha n}(d_n))) = r_1(d'(r_{\alpha 1}d_1)\ldots(r_{\alpha n}d_n))$$
$$= dd_1\ldots d_n$$

hence by extensionality $g_\beta(F) = d$.

If $g_\beta(F) = d$ we shall say that d is the graph of F. It follows that F is S-computable if and only if the graph of F is in Q_0.

5.3 We shall prove closure under substitution in the following form. Let H be a functional of type (α,β) G be a functional of type $\beta' = (\alpha 1,\ldots,\alpha n,\alpha)$. Now we may define a functional F of type β by

$$F(x_1,\ldots,x_n) = H(G(x_1,\ldots,x_n),x_1,\ldots,x_n)$$

Let d_0 be the graph of H, and d' be the graph of G, and assume both are in the basis Q. Then there is $d \in Q$ such that for arbitrary $d_1,\ldots,d_n$ the following relation holds: $dd_1\ldots d_n = d_0(d'd_1\ldots d_n)d_1\ldots d_n$. Hence it follows from Theorem 5.2 i) that

$$G(h_{\alpha 1}(d_1),\ldots,h_{\alpha n}(d_n)) = h_\alpha(d'd_1\ldots d_n)$$
$$g_1(F(h_{\alpha 1}(d_1),\ldots,h_{\alpha n}(d_n)) =$$
$$= g_1(H(h_\alpha(d'd_1\ldots d_n),h_{\alpha 1}(d_1),\ldots,h_{\alpha n}(d_n)))$$
$$= d_0(d'd_1\ldots d_n)d_1\ldots d_n$$
$$= dd_1\ldots d_n$$

hence d is the graph of F so F is Q-definable.

5.4 Definitions by recursion are introduced via a functional F of type $(\alpha 1,\ldots,\alpha n,\alpha,\alpha)$. We know there is a functional G of type $\beta' = (\alpha 1,\ldots,\alpha n,\alpha)$ such that the following identity holds:

$$F(x_1,\ldots,x_n,G(x_1,\ldots,x_n),y_1,\ldots,y_k) = G(x_1,\ldots,x_n,y_1,\ldots y_k)$$

We shall show that in case F is Q-definable then Q is also Q-definable.

The proof uses the term Y introduced in 2.4. We put $Va(Y) = R \in Q_0$. Since Yv_1 conv $v_1(Yv_1)$ it follows that for any $d \in D$, the relation $Rd = d(Rd)$ holds. Furthermore by inspecting the terms Y such that Yv_1 red Y it follows that $Rd = \cup\{d^n\bot : n \geq 0\}$. Hence if d' is any element of D such that $dd' \subset d'$ it follows that for each $n \geq 0$, $d^n\bot \subset d'$ hence $Rd \subset d'$.

Now let d be the graph of the functional F, and take d_0 such that $d_0d_1\ldots d_n = R(dd_1\ldots d_n)$. Hence using Theorem 5.2 we can prove the following properties of d_0:

i) $d_0(r_{\alpha 1}d_1)\ldots(r_{\alpha n}d_n) = d_0d_1\ldots d_n$

ii) $r_\alpha(d_0d_1\ldots d_n) = d_0d_1\ldots d_n$

iii) $d_0(r_{\alpha 1}d_1)\ldots(r_{\alpha n}d_n)(r_{11}d_{n+1})\ldots(r_{1k}d_{n+k}) = d_0d_1\ldots d_nd_{n+1}\ldots d_{n}$

iv) $r_j(d_0d_1\ldots d_nd_{n+1}\ldots d_{n+k}) = d_0d_1\ldots d_nd_{n+1}\ldots d_{n+k}$

Let G be the functional of type β' defined by d_0. From the above properties it follows that d_0 is actually the graph of G, hence Theorem 5.2 applies. Now we put $d_m = g_{\alpha m}(x_m)$, $m = 1,\ldots,n$ and $d_{n+t} = g_{1t}(y_t)$, $t = 1,\ldots,k$. We compute as follows, using that $G(x_1,\ldots,x_n) = h_\alpha(d_0d_1\ldots d_n)$:

$$F(x_1,\ldots,x_n,G(x_1,\ldots,x_n),y_1,\ldots,y_k) =$$
$$= h_j(dd_1\ldots d_n(d_0d_1\ldots d_n)d_{n+1}\ldots d_{n+k})$$
$$= h_j(d_0d_1\ldots d_nd_{n+1}\ldots d_{n+k})$$
$$= G(x_1,\ldots,x_n,y_1,\ldots,y_k)$$

We must also prove that the functional G is minimal. Assume G' is another functional of type β' such that for arbitrary $x_1,\ldots,x_n$:

$$F(x_1,\ldots,x_n,G'(x_1,\ldots,x_n)) \subset_\alpha G'(x_1,\ldots,x_n)$$

If d' is the graph of G' then for arbitrary $d_1,\ldots,d_n$

$$dd_1\ldots d_n(d'd_1\ldots d_n) \subset d'd_1\ldots d_n$$

hence from the properties of R and the definition of d_0

$$d_0d_1\ldots d_n \subset d'd_1\ldots d_n$$

but this implies $d_0 \subset d'$, hence $G = h_{\beta'}(d_0) \subset_{\beta'} h_{\beta'}(d') = G'$

6. DETERMINISTIC COMPUTABILITY

6.1 The notion of computability presented in this paper is essentially relative. If S = (H,C,Int) is an algorithmic system and F is a S-computable functional, then we can reduce the evaluation of F to partial evaluations using the basic operators in S, i.e. those operators of the form int(c) for $c \in C$. Whether these basic operators are computable in some absolute sense, is a question that we do not intend to discuss here.

In this section we want to discuss a more restricted question, namely deterministic computability. An operator is deterministically computable in case the evaluation is given by an effective procedure with a terminating machinery, such that whenever the procedure terminates an output value is produced, and in case the procedure does not terminate the output is undefined. The general procedure described in secion 2 to evaluate $Ev_1(X)(\sigma)$ is not deterministic, even if there is some Y such that X red Y and $Ev_1(X)(\sigma) = Ev_1(Y^p)(\sigma)$. For there is no rule to detect this situation so the evaluation must proceed to generate all terms Y such that X red Y. In other words no terminating machinery is available.

6.2 As before the letter α denotes a ground operator (i1,...,ik,j). A ground operator f of type α is *regular* in case the following condition is satisfied: If $f(x_1,\dots,x_k) = e$ and $e \neq \bot_j$, and $x_1',\dots,x_k'$ are values such that either $x_m = \bot_{im}$ or $x_m = x_m'$, $m = 1,\dots,k$, then $f(x_1',\dots,x_k') = e$.

We say that a system $S = (H,C,\mathrm{int})$ is regular if $\mathrm{int}(c)$ is a regular operator for every $c \in C$. An assignment σ is regular if $\sigma(v)$ is a regular operator for every variable v.

Lemma 6.1. Let S be a regular system, and σ and σ' be regular assignments, such that for every variable v, either $\sigma(v) = \bot_\alpha$ for some type α, or $\sigma(v) = \sigma'(v)$. If X and Y are terms in normal form such that X Ω-cov Y holds, and $Ev_1(X)(\sigma) = e$ where $e \neq \bot_1$, then $Ev_1(Y)(\sigma') = e$.

Proof by induction on the structure of X with cases arising from the rules defining the covering relation.

Let $X \equiv X_0X_1 \ldots X_n$ where X_0 is atomic. Since $e \neq \bot_1$ rule CV2 applies and $e = f(e_1, \ldots, e_k)$ for some ground operator f of type α. But then $Ev_1(Y)(\sigma') = f(e_1', \ldots, e_k') = e'$, where by the induction hypothesis either $e_m = \bot_{1m}$ or $e_m = e_m'$, $m = 1, \ldots, k$. Since f is regular we have $e = e'$.

The other cases follow easily from the induction hypothesis.

Theorem 6.1. Let S be a regular system, and σ and σ' be regular assignments, such that for every variable v, either $\sigma(v) = \bot_\alpha$ for some type α, or $\sigma(v) = \sigma'(v)$. If X is any terms such that $Ev_1(X)(\sigma) = e$ where $e \neq \bot_1$ then $Ev_1(X)(\sigma') = e$.

We put $Ev_1(X)(\sigma') = e'$. Then $e \subset_1 e'$ follows from Lemma 6.1 applied to Y^p whenever X red Y. To prove $e' \subset_1 e$ take any Y such that X red Y. By the CHRP there is Z such that Y red Z and $Ev_1(Z^p)(\sigma) = e^*$ where $e^* \neq \perp_1$ and $e^* \subset_1 e$. Then by Lemma 6.1 $Ev_1(Z^p)(\sigma') = e^*$, hence $Ev_1(Y^p)(\sigma') \subset_1 e^*$.

Corollary. If S is a regular system and f is S-computable ground operator, then f is regular.

Immediately from the Theorem noting that the elements of any domain H_j are regular.

6.3 We shall say that a system S is *partial evaluation deterministic* if whenever X is a term and σ is some regular assignment such that $Ev_1(X^p)(\sigma) = e$ and $e \neq \perp_1$, then $Ev_1(X)(\sigma) = e$. If S is partial evaluation deterministic then there is a deterministic procedure to evaluate $Ev_1(X)(\sigma)$ for any term X and regular assignment σ. Simply generate the terms Y such that S red Y and evaluate $Ev_1(Y^p)(\sigma) = e$. If for some Y the value e is different from $\perp_1$ stop and output is e. Otherwise the output is $\perp_1$.

Theorem 6.2. A system S is partial evaluation deterministic if and only if it is regular.

First assume that S is partial evaluation deterministic and put $int(c) = f$ for $c \in C$, where f is a ground operator of type α, and we may assume $k \geq 1$. Assume that $f(x_1,\ldots,x_k) = e$ where $e \neq \perp_1$. Now put $I \equiv \lambda v_1 v_1$, and for $m = 1,\ldots,k$ take x'_m such that either $x_m = \perp_{1m}$ or $x'_m = x_m$, $X_m \equiv Iv_m$, $\sigma(v_m) = x'_m$. Put $X \equiv cX_1\ldots X_k$. Then X red $cv_1\ldots v_m$, hence $Ev_j(X)(\sigma) = f(x'_1,\ldots,x'_k) = e'$. On the other hand X red $cY_1\ldots Y_k \equiv Y$ where $Y_m \equiv v_m$ if $x_m \neq \perp_{1m}$ and $Y_m \equiv X_m$ otherwise. Now $Ev_j(Y^p)(\sigma) = f(x_1,\ldots,x_k) = e$. Since S is partial evaluation deterministic we have $e = e'$, so f is a regular ground operator.

Assume now that the system S is regular and X is a term such that $Ev_i(X^p)(\sigma) = e$ where $e \neq \perp_i$ and σ is a regular assignment. Then by Lemma 6.1 if Y is a term such that X red Y holds, we have $Ev_i(Y^p)(\sigma) = e$, hence $Ev_i(X)(\sigma) = e$.

It is possible to argue that any deterministically computable operator must be regular. For assume f is a ground operator of type α, and there is a deterministic procedure to evaluate $f(x_1,\ldots,x_k) = e$ where $e \neq \perp_j$ and $x_m = \perp_{im}$. Since the input x_m may be given via a deterministic but infinite computation it follows that the evaluation of e must ignore the input x_m, hence the same output will result if x_m is replaced by x'_m.

6.4 We conclude this section with some considerations on the role of the overdefined element. We are interested in some conditions imposing restrictions on the manner the overdefined element determines the output. For instance we may want that whenever some input is overdefined, and it is not ignored by the computation, then the output should be also overdefined. We may go further and require that the output should not be overdefined unless some input is overdefined. These are consistency conditions, in the sense that any inconsistency in the output is totally determined by inconsistencies in the inputs.

Let f be a ground operator of type α. We say that f is *weakly consistent* if the following condition is satisfied: If $f(x_1,\ldots,x_k) = e$ and $e \neq \tau_j$, and $x'_1,\ldots,x'_k$ are values such that $x_m = \tau_{im}$ or $x_m = x'_m$, $m = 1,\ldots,k$, then $f(x'_1,\ldots,x'_k) = e$. And we say that f is *consistent* if it is weakly consistent and whenever $f(x_1,\ldots,x_k) = \tau_j$ then for some m, $x_m = \tau_{im}$.

A system S is (*weakly*) *consistent* if int(c) is (weakly) consistent for every $c \in C$. An assignment σ is (*weakly*) *consistent* if $\sigma(v)$ is (weakly) consistent for every variable v. Note that any element in H_j is weakly consistent, but only the elements different from τ_j are consistent.

Lemma 6.2. Let S be a weakly consistent system, and σ and σ' be weakly consistent assignments such that for every variable v, either $\sigma(v) = \tau_\alpha$ for some type α, or $\sigma(v) = \sigma'(v)$. If X is a term in normal form such that $Ev_i(X)(\sigma) = e$ where $e \neq \tau_i$ then $Ev_i(X)(\sigma') = e$. Furthermore if S and σ are consistent then $Ev_i(X)(\sigma) \neq \tau_i$.

The proof is by induction on the structure of X, similar to the proof of Lemma 6.1.

Theorem 6.3. Let S be a weakly consistent system, and σ and σ' be weakly consistent assignments such that for every variable v, either $\sigma(v) = \tau_\alpha$ for some type α, or $\sigma(v) = \sigma'(v)$, If X is any term such that $Ev_i(X)(\sigma) = e$ where $e \neq \tau_i$ then $Ev_i(X)(\sigma') = e$. Furthermore if both X and σ are consistent then $Ev_i(X)(\sigma) \neq \tau_i$.

The first part is trivial since from Lemma 6.2 it follows that whenever X red Y then $Ev_1(Y^p)(\sigma) = Ev_1(Y^p)(\sigma')$. To prove the second part note that if $e = \perp_1$ we are done, and if $e \neq \perp_1$ then by Theorem 6.2 there is Y such that X red Y and $Ev_1(Y^p)(\sigma) = e$. Hence by Lemma 6.2 we have $e \neq \top_1$

Corollary. If S is regular and consistent and f is S-computable, then f is consistent.

7. EXAMPLES

7.1 We shall consider here a number of algorithmic systems, showing their power and limitations. It is convenient to distinguish different basic structures using superscripts. So in general a given algorithmic system is of the form $S_j = (H^j, C_j, \mathrm{int}_j)$ where $j \in \omega$. The domains in the structure H^j are denoted as H^j_i where $i \in \omega$. It is not necessary to define H^j_i for all i. To avoid ambiguities we agree that in case it is not explicitly defined then $H^j_i = \{\bot, \top\}$.

Let A be some set and B another set contained in some domain D. Let $f(x_1,\ldots,x_k)$, $k \geq 1$, be some partial function defined on A with values in B. The function is partial so it may be undefined for some arguments. Suppose now that in some basic structure we have $H_i = A^+$ and $H_j = D$. We want to extend f to a ground operator f* of type $(i1,\ldots,ik,j)$ where $i1 = \ldots = ik = i$. If $f(x_1,\ldots,x_k) = e$ is defined we put $f^*(x_1,\ldots,x_k) = e$. If $f(x_1,\ldots,x_k)$ is undefined, or some $x_m = \bot_i$, we put $f^*(x_1,\ldots,x_k) = \bot_j$. In all the other case we put $f^*(x_1,\ldots,x_k) = \top_j$. Note that f* is a regular and consistent ground operator.

In proving a functional to be S-computable we shall use the results of section 5 on definability. In general functionals will be defined using substitution and recursion. Note that in case a functional F is S-computable and F' is a functional defined by adding new variables, and also permutation of variables, then F' is S-computable. For suppose the close term X computes in S the functional $F(x_1,x_2,x_3)$. Then the term $Y \equiv \lambda v_1 \lambda v_2 \lambda v_3 \lambda v_4 (X v_4 v_2 v_1)$ computes the functional $F'(x_1,x_2,x_3,x_4) = F(x_4,x_2,x_1)$. With this understanding it should be clear that the substitution rule proved in 5.3 can be used for any kind of substitution, and computability will be preserved.

7.2 The first example is the system $S_1 = (H^1, C_1, \mathrm{int}_1)$ where we put $H_1^1 = \omega^+$.(Recall that H_0 is always Bool). We put $C_1 = \{0,s,p,b,c\}$(we do not mention explicitly the constant tt and ff which are assumed to be in any system) The function int_1 is defined as follows:

i) $\mathrm{int}_1(0) = 0 \in \omega^+$

ii) $\mathrm{int}_1(s) = s^*$ of type (ι,ι) where s is the function on ω defined by $s(n) = n + 1$.

iii) $\mathrm{int}_1(p) = p^*$ of type (ι,ι) where p is the function on ω defined by $p(0) = 0$ and $p(n+1) = n$.

iv) $\mathrm{int}_1(b) = b^*$ of type $(\iota,0)$ where b is the function defined on ω by $b(0) =$ true and $b(n+1) =$ false.

v) $\mathrm{int}_1(c) = c^*$ of type (ι,ι,ι) where c is the function defined on ω by $c(n,m) = 0$.

We shall show that whenever f is a partial recursive function on ω, then f^* is S_1-computable. The ideas are well known and we just outline the argument.

First note that if $c_k(x_1,\ldots,x_k) = 0$ is the constant zero function with k arguments, the c_k^* can be defined from c^* by substitution. If $f(x_1,\ldots,x_k) = x_i$ is some identity function, the f^* is not an identity function in ω^+. But we can define f^* as follows:

$$f^*(x_1,\ldots,x_k) = \text{Cond}\,(b^*(c_k^*(x_1,\ldots,x_k)),x_1,x_1)$$

Now consider a total numerical function defined by primitive recursion from total functions in the form

$$f(x_1,\ldots,x_k,0) = f_1(x_1,\ldots,x_k)$$
$$f(x_1,\ldots,x_k,y+1) = f_2(x_1,\ldots,x_k,y,f(x_1,\ldots,x_k,y))$$

To simplify the notation we put $k = 1$ and define an operator d of type $(1,1,1)$ as follows

$$d(x,y) = \text{Cond}_1(b^*(c^*(x,y)),b^*(y),b^*(y))$$

Next we define a functional F of type $(1,(1,1),1,1)$ by

$$F(x,z,y) = \text{Cond}_1(d(x,y),f_1^*(x),f_2^*(x,p^*(y),z(y)))$$

Now let f_0 be the recursive solution of z in the functional F, hence

$$F(x,f_0(x),y) = f_0(x,y)$$

From this it follows easily that $f^* = f_0$.

Finally we consider a partial recursive function defined by minimization from a total recursive function in the form

$$f(x_1,\ldots,x_k) = \mu y[f_1(x_1,\ldots,x_k,y) = 0]$$

Again we assume $k = 1$ and define a functional F of type $(1,(1,1),1,1)$ by

$$F(x,z,y) = \text{Cond}_1(b^*(f_1^*(x,y)),0,z(y+1)+1)$$

Let f_0 of type $(1,1,1)$ be the recursive solution for z, hence $F(x,f_0(x),y) = f_0(x,y)$, and f_0 is minimal with this property. We can show now that $f^*(x) = f_0(x,0)$.

First note that if $x = \perp_1$ or $x = \top_1$ the definition is consistent with the properties of f^*. Take $x \in \omega$; then we can prove for any $n \in \omega$,that if m is the least number, $m \geq n$ such that $f_1(x,m) = 0$, then $f_0(x,n) = m - n$. The proof is by induction on $m - n$. Finally note that if $f_1(x,y) \neq 0$ then $F(x,z,y) = z(y+1) + 1$. Hence if $f_1(x,y) \neq 0$ for all $y \in \omega$, then for any $m \geq 0$ and with $\alpha = (1,1)$ we have for any $y \in \omega$:

$$F(x)^{m+1}(\perp_\alpha)(y) = F(x,F(x)^m(\perp_\alpha),y) = \perp_1$$

hence in case $f(x)$ is undefined then $f_0(x,0) = \perp_1$.

The system S_1 appears to be reasonably complete. Still we note that there are operators which are intuitively computable, but are not S_1-computable. For instance the operator f of type (1,1,1) defined by $f(x,y) =$ the meet of x and y in ω^+, is not consistent so it is not S_1-computable. We may want to extend S_1 to allow this and other similar operators to be computable. Or we may prefer to ignore the situation on the assumption that the introduction of the overdefined element in this context is artificial and does not correspond to any real computing problem. In fact as long as one is interested only in deterministic computability it seems more reasonable to eliminate altogether the overdefined element. We shall discuss this possibility in the next section.

As we may expect the system S_1 is defficient in terms of nondeterministic computability. For example let g be a partial numerical function on ω. We want to compute for each $x \in \omega$ the unique y such that $g(y) = x$, whenever such y exists. In case there is no y such that $g(y) = x$ the output is undefined. In case there are two or more the output is overdefined. We formalize this in a functional F of type $((\iota,\iota),\iota,\iota)$ which is defined

$$F(g,x) = \cup\{\mathrm{Cond}_1(E^*(g(y),x),y,\bot_1) : y \in \omega\}$$

where E is the equality predicate on ω, so E^* if of type $(\iota,\iota,0)$.

It is easy to show that F is not S_1-computable. For assume that X is a closed term that computes F. Define $g_1(y) = \bot_1$ in case $y = \bot_1$ or $y \in \omega$ and $y \neq 0$, $g_1(0) = 0$ and $g_1(\top_1) = \top_1$. Then $F(g_1,0) = 0$ so there is a term Y such that Xv_1v_2 red Y and $Ev_1(Y^p)(\sigma) = 0$, where $\sigma(v_1) = g_1$ and $\sigma(v_2) = 0$. But the evaluation of Y^p uses only a finite number of values of g_1. Let n be such that $g_1(n)$ is not used. Now define $g_2(n) = 0$ and otherwise $g_2(y) = g_1(y)$. Now we have $F(g_2,0) = \top_1$. On the other hand $Ev_1(Y^p)(\sigma') = 0$ where $\sigma'(v_1) = g_2$ and $\sigma'(v_2) = 0$ and this contradicts Theorem 6.3.

7.3 We consider now a system $S_2 = (H^2,C_2,\mathrm{int}_2)$ which is an extension of S_1 and it is not regular. We take $H^2 = H^1$, and $C_2 = C_1 \cup \{j\}$. We define int_2 the same as int_1 on C_1, and put $\mathrm{int}_2(j) = \jmath$ of type (ι,ι,ι) where $\jmath(x,y)$ = the join of x and y in ω^+.

In the system S_2 we may compute truly nondeterministic functionals. For instance the functional F defined above is S_2-computable. To show this define a functional H of type ((1,1),1,(1,1),1,1) as follows

$$H(g,x,h,y) = j(\mathrm{Cond}_1(E^*(g(y),x),y,\bot_1),h(y+1))$$

If h_0 is the recursive solution for h, it is easy to verify that $F(g,x) = h_0(g,x,0)$.

The example S_3 is of the form (H^3,C_3,int_3) and it is also an extension of S_1. We put $H^3 = \omega^+$, $H_2^3 = P\omega$ = the power set of ω with the usual inclusion ordering. We use greek letters γ and δ to denote elements of $P\omega$. We put $C_3 = C_1 \cup \{J,sg,ep\}$. The function int_3 is equal to int_1 on C_1. We put $\mathrm{int}_3(J) = J$ of type (2,2,2) where $J(\gamma,\delta)$ = the join or union of γ and δ in $P\omega$; $\mathrm{int}_3(sg) = sg^*$ of type (1,2) where $sg(x) = \{x\}$; $\mathrm{int}_3(ep) = ep$ of type (1,2,0) where $ep(x,\gamma) = \bot_0$ in case $x = \bot_1$ or $x \in \omega$ and $x \notin \gamma$, $ep(x,\gamma)$ = true in case $x \in \gamma$, and $ep(\tau_1,\gamma) = \tau_0$.

In S_3 we can define a functional F of type ((1,1),(1,2),1,2) as follows

$$F(g,H,y) = J(sg^*(g(y)),H(y+1))$$

Let H_0 be the recursive solution for H, so we have

$$F(g,H_0(g),y) = H_0(g,y)$$

It is easy to see that if g is any partial numerical function on ω, then $H_0(g^*,0)$ = the range of g. It follows that all recursively enumerable sets are S_3-computable.

On the other hand we can define in S_3 an extension of the operator ep. Let $e_0, e_1, \ldots$ be the canonical enumeration of all finite subsets of ω (see [7] page 525). Hence for $n \in \omega$, e_n is a finite subset of ω and every finite subset can be expressed in this way. We define an operator in of type $(\iota,2,0)$ as follows: $in(x,\gamma) = \bot_0$ in case $x = \bot_1$ or $x \in \omega$ and e_x is not a subset of γ; $in(x,\gamma) =$ true in case $x \in \omega$ and e_x is a subset of γ; $in(\tau_1,\gamma) = \tau_0$. The operator in is S_3-computable. The details are left to the reader.

Now let assume some recursive pairing of ω, of the form $pr(x,y) = z$ with inverse functions $ls(z) = x$ and $rs(z) = y$. Consider the following functional G of type $(2,2,\iota,2)$

$$G(\gamma,\delta,y) =$$

$$sg^*(Cond_1(ep(y,\gamma),Cond_1(in(ls^*(y),\delta),rs^*(y),\bot_1),\bot_1))$$

Now define a functional H of type $(2,2,(\iota,\iota),\iota,\iota)$

$$H(\gamma,\delta,h,y) = J(G(\gamma,\delta,y),\ h(y+\iota))$$

Let h_0 be the recursive solution for h and define

$$Ap(\gamma,\delta) = h_0(\gamma,\delta,0)$$

The operator Ap of type $(2,2,2)$ is actually the application operator of the graph model, denoted as **fun** in [7] page 526 (see also [4]).

Let recall that enumeration operators are of type $(2,2)$ and are defined by fixing the first argument of Ap to some recursively enumerable set. It follows that all the enumeration operators are S_3-computable.

8. CONCLUSIONS

8.1 The constructions presented in this paper involve three different elements: the language BAL, the basic structure H, and the reflexive domain D. The connection between BAL and H is given by the function Ev_1; and H is related to D by some embedding. There is another connection between BAL and D via the function Va, that commutes with the others in case the embedding is minimal. The existence of this factorization depends strongly on continuity assumptions. Apparently as long as continuity is enforced there is no need to pay attention to the nature of the objects in the basic structure. They may be numbers or sets, finite or infinite, the factorization is always available and makes it possible to interpret the programs in BAL as real mathematical expressions.

Continuity seems to determine the limit of the method. If we allow monotonic but not continuous operators, we get operators that are closed neither under substitution nor under recursion. To deal with monotonic operators we need a notion of transfinite reduction which is far beyond the finitary rules of BAL.

8.2 The theory is developed under the assumption that domains contain a top overdefined element. A motivation for this was suggested in the introduction. But an alternative construction is possible without enforcing such restrictions We may redefine the notion of domain, requiring a least but not a greatest element. Then we must redefine the construction A^+ which now is obtained by adding only a least element. This changes the definition of Bool and also of $\underset{j\in J}{\oplus} D_j$. With these changes it can be checked that all definitions,proofs and constructions in the paper not involving a top element are still valid. In particular the minimal embedding will result in a domain D without top element, but otherwise satisfying all the required conditions.

This alternative theory without overdefined elements may present some interest as a generalization of deterministic computability. But to study nondeterministic computations the overdefined element seems to be essential. In fact there are reasons that suggest that the best generalization would be to assume all the domains to be complete lattices (as happens to be the case in all the examples considered).

8.3 The results on deterministic computability suggest other alternatives. In a regular system there is no need to define the function Ev_1 as a limit, and hence the domain structure is not necessary. We need only the existence of a bottom element, and monotonicity only relative to this element. We conjecture that a theory along these lines will be equivalent to Moschovakis's abstract computability.

Several directions may be suggested to continue this investigation. We do not think that the ultimate significance of the minimal embedding is expressed by the applications in this paper. The reflexive domain seems to provide a mathematical structure for the apparently chaotic world of terms and reduction in BAL. We may expect eventually to find rules for the safe manipulation and combinations of programs. This application will certainly be of interest in practical computer science.

In another direction there is obvious interest in studying more examples, to determine the real scope of the theory. And there is interest also in the study of higher order structures, not only higher order functionals, but also structures defined by recursion.

REFERENCES

[1] J.R.HINDLEY, B. LERCHER, and J.P. SELDIN, Introduction to Combinatory Logic, Cambridge University Press, London, 1972.

[2] H. ROGERS, Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.

[3] Y.N. MOSCHOVAKIS, Abstract First Order Computability I, Trans. Amer.Math.Soc., vol.138, pp.427-464,1969.

[4] L.E. SANCHIS, Reducibilities in two Models for Combinatory Logic, The Journal of Symbolic Logic, vol.44, pp.221-234, 1979.

[5] D. SCOTT, Continuous Lattices, Proc. 1971 Dalhousie Conference, in Lecture Notes in Mathematics vol.274, pp.97-136, Springer-Verlag, New York, 1972.

[6] D. SCOTT, Outline of a Mathematical Theory of Computation, Proc. 4th Ann.Princeton Conf. on Information Sciences and Systems,pp.169-176, 1970.

[7] D. SCOTT, Data Types as Lattices, SIAM Journal of Computing, vol.5, pp.522-587, 1976.

[8] C.P. WADSWORTH, The Relation between Computational and Denotational Properties for Scott's D_∞-Models of the Lambda-Calculus, SIAM Journal of Computing, vol.5, pp.448-521, 1976.

[9] E.G. WAGNER, Uniformly Reflexive Structures: on the Nature of Gödelization and Relative Computability, Trans.Amer.Math.Soc., vol.14, pp.1-41, 1969.